Project Clearwater

Release 1.0

Sep 14, 2017

Overview

Project Clearwater is an open-source IMS core, developed by Metaswitch Networks and released under the GNU GPLv3. You can find more information about it on our website.

CHAPTER 1

Latest Release

The latest stable release of Clearwater is "Nazgûl".

CHAPTER 2

Architecture

Clearwater is architected from the ground up to be fully horizontally scalable, modular and to support running in virtualized environments. See our Clearwater Architecture page for a bird's eye view of a Clearwater deployment and a guided tour of the various functional components that comprise it.

CHAPTER 3

Getting Started

- Installation Instructions
- Making your first call
- Running the live tests
- A tour of Clearwater

CHAPTER 4

Looking Deeper

To look at the optional extra features and function of your Clearwater deployment and for discussions about Clearwater scaling and redundancy, see Exploring Clearwater.

CHAPTER 5

Getting Source Code

All the source code is on GitHub, in the following repositories (and their submodules).

- chef - Chef recipes for Clearwater deployment
- clearwater-infrastructure - General infrastructure for Clearwater deployments
- clearwater-logging - Logging infrastructure for Clearwater deployments
- clearwater-live-test - Live test for Clearwater deployments
- clearwater-readthedocs - This documentation repository
- crest - RESTful HTTP service built on Cassandra - provides Homer (the Clearwater XDMS) and Homestead-prov (the Clearwater provisioning backend)
- ellis - Clearwater provisioning server
- sprout - Sprout and Bono, the Clearwater SIP router and edge proxy
- homestead - Homestead, the Clearwater HSS-cache.
- ralf - Ralf, the Clearwater CTF.

CHAPTER 6

Contributing

You can contribute by making a GitHub pull request. See our documented Pull request process.

If you want to contribute specifically to this documentation (e.g. to fix a typo or document a missing option), the Markdown files for it are at https://github.com/Metaswitch/clearwater-readthedocs.

There is more information about contributing to Project Clearwater on the community page of our project website.

CHAPTER 7

Help

If you want help, or want to help others by making Clearwater better, see the Support page.

CHAPTER 8

License and Acknowledgements

Clearwater's license is documented in LICENSE.txt.

It uses other open-source components as acknowledged in README.txt.

Clearwater Architecture

Clearwater was designed from the ground up to be optimized for deployment in virtualized and cloud environments. It leans heavily on established design patterns for building and deploying massively scalable web applications, adapting these design patterns to fit the constraints of SIP and IMS. The Clearwater architecture therefore has some similarities to the traditional IMS architecture but is not identical.

In particular ...

- All components are horizontally scalable using simple, stateless load-balancing.
- All long lived state is stored on dedicated "Vellum" nodes which make use of cloud-optimized storage technologies such as Cassandra. No long lived state is stored on other production nodes, making it quick and easy to dynamically scale the clusters and minimizing the impact if a node is lost.
- Interfaces between the front-end SIP components and the back-end services use RESTful web services interfaces.
- Interfaces between the various components use connection pooling with statistical recycling of connections to ensure load is spread evenly as nodes are added and removed from each layer.

The following diagram illustrates the Clearwater architecture and its components.

Bono (Edge Proxy)

The Bono nodes form a horizontally scalable SIP edge proxy providing both a SIP IMS Gm compliant interface and a WebRTC interface to clients. Client connections are load balanced across the nodes. The Bono node provides the anchor point for the client's connection to the Clearwater system, including support for various NAT traversal

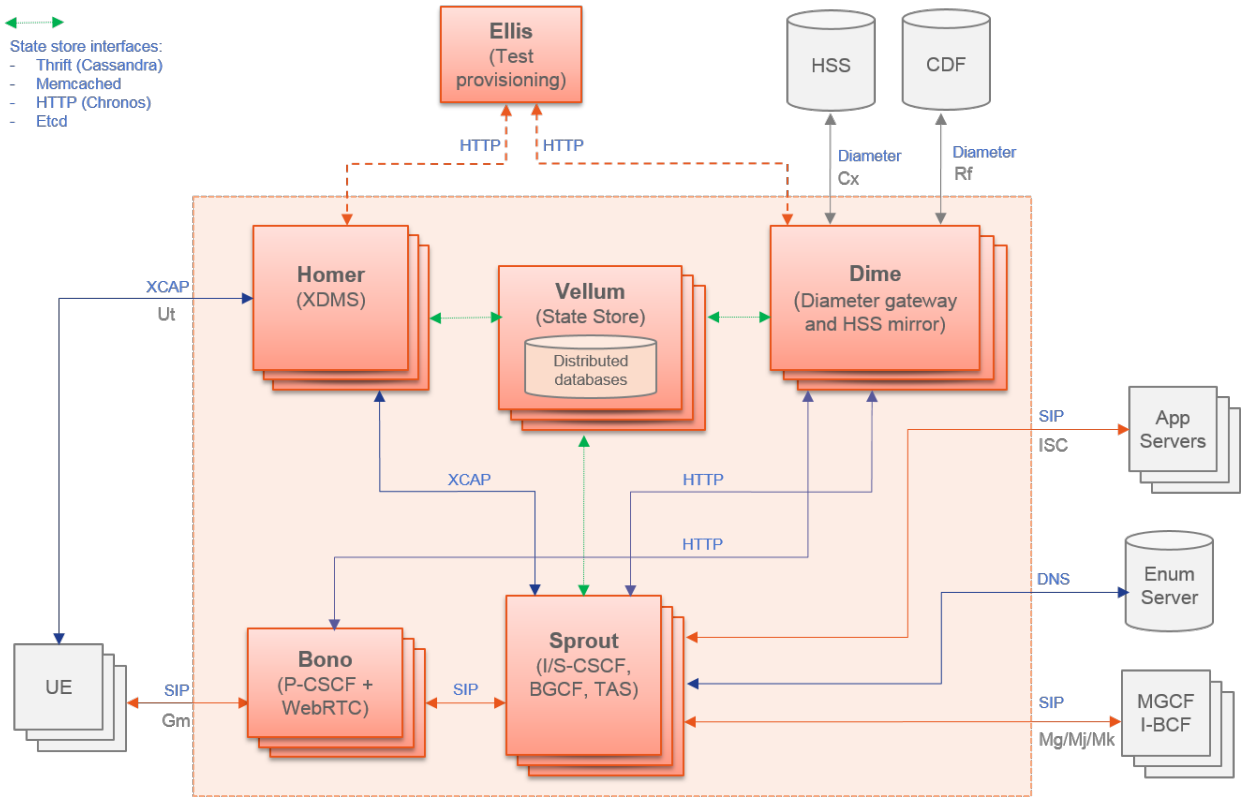

Fig. 8.1: Architecture

mechanisms. A client is therefore anchored to a particular Bono node for the duration of its registration, but can move to another Bono node if the connection or client fails.

Clients can connect to Bono using SIP/UDP or SIP/TCP. Bono supports any WebRTC client that performs call setup signaling using SIP over WebSocket.

Alternatively, Clearwater can be deployed with a third party P-CSCF or Session Border Controller implementing P-CSCF. In this case Bono nodes are not required.

Sprout (SIP Router)

The Sprout nodes act as a horizontally scalable, combined SIP registrar and authoritative routing proxy, and handle client authentication and the ISC interface to application servers. The Sprout nodes also contain the in-built MMTEL application server. SIP transactions are load balanced across the Sprout cluster, so there is no long-lived association between a client and a particular Sprout node. Sprout does not store any long-lived data itself and instead uses - web services interfaces to Homestead and Homer to retrieve HSS configuration such as authentication data/user profiles and MMTEL service settings - APIs to Vellum for storing subscriber registration data and for running timers.

Sprout is where the bulk of the I-CSCF and S-CSCF function resides, with the remainder provided by Dime (and backed by the long-lived data stores on Vellum).

Dime (Diameter gateway)

Dime nodes run Clearwater's Homestead and Ralf components.

Homestead (HSS Cache)

Homestead provides a web services interface to Sprout for retrieving authentication credentials and user profile information. It can either master the data (in which case it exposes a web services provisioning interface) or can pull the data from an IMS compliant HSS over the Cx interface. The Homestead nodes themselves are stateless - the mastered / cached subscriber data is all stored on Vellum (Cassandra for the mastered data, and Astaire/Memcached for the cached data).

In the IMS architecture, the HSS mirror function is considered to be part of the I-CSCF and S-CSCF components, so in Clearwater I-CSCF and S-CSCF function is implemented with a combination of Sprout and Dime clusters.

Ralf (CTF)

Ralf provides an HTTP API that both Bono and Sprout can use to report billable events that should be passed to the CDF (Charging Data Function) over the Rf billing interface. Ralf is stateless, using Vellum to maintain the long lived session state and run the timers necessary to enable it to conform to the Rf protocol.

Vellum (State store)

As described above, Vellum is used to maintain all long-lived state in the dedployment. It does this by running a number of cloud optimized, distributed storage clusters. - Cassandra. Cassandra is used by Homestead to store authentication credentials and profile information when an HSS is not in use, and is used by Homer to store MMTEL service settings. Vellum exposes Cassandra's Thrift API. - etcd. etcd is used by Vellum itself to share clustering information between Vellum nodes and by other nodes in the deployment for shared configuration. - Chronos. Chronos is a distributed, redundant, reliable timer service developed by Clearwater. It is used by Sprout and Ralf nodes to enable timers to be run (e.g. for SIP Registration expiry) without pinning operations to a specific node (one node can set the timer and another act on it when it pops). Chronos is accessed via an HTTP API. - Memcached / Astaire. Vellum also runs a Memcached cluster fronted by Astaire. Astaire is a service developed by Clearwater that enabled more rapid scale up and scale down of memcached clusters. This cluster is used by Sprout for storing registration state, Ralf for storing session state and Homestead for storing cached subscriber data.

Homer (XDMS)

Homer is a standard XDMS used to store MMTEL service settings documents for each user of the system. Documents are created, read, updated and deleted using a standard XCAP interface. As with Homestead, the Homer nodes use Vellum as the data store for all long lived data.

Ellis

Ellis is a sample provisioning portal providing self sign-up, password management, line management and control of MMTEL service settings. It is not intended to be a part of production Clearwater deployments (it is not easy to horizontally scale because of the MySQL underpinnings for one thing) but to make the system easy to use out of the box.

Load Balancing

In a cloud scalable system like Clearwater load balancing is an important part of making the system horizontally scale in a robust way. Clearwater uses a variation on DNS load balancing to ensure even loading when clusters are being elastically resized to adapt to changes in total load.

As an example, a single domain name is configured for all the Sprout nodes. Each Bono node maintains a pool of SIP connections to the Sprout nodes, with the target node for each connection selected at random from the list of addresses returned by DNS. Bono selects a connection at random for each SIP transaction forwarded to Sprout. The connections in the pool are recycled on failure and periodically, selecting a different address from the list returned by the DNS server each time.

A similar technique is used for the HTTP connections between Sprout and Homer/Dime - each Sprout maintains a pool of connections load balanced across the Homer/Dime clusters and periodically forces these connections to be recycled.

Reliability and Redundancy

Traditional telco products achieve reliability using low-level data replication, often in a one-to-one design. This is both complex and costly - and does not adapt well to a virtualized/cloud environment.

The Clearwater approach to reliability is to follow common design patterns for scalable web services - keeping most components stateless and storing long-lived state in specially designed reliable and scalable clustered data stores.

Both Bono and Sprout operate as transaction-stateful rather than dialog-stateful proxies - transaction state is typically short-lived compared to dialog state. As the anchor point for client connections for NAT traversal, the Bono node used remains on the signaling path for the duration of a SIP dialog. Any individual Sprout node is only in the signaling path for the initial transaction, and subsequent requests are routed through the entire Sprout cluster, so failure of a Sprout node does not cause established SIP dialogs to fail. Long-lived SIP state such as registration data and event subscription state is stored in a clustered, redundant shared data store (memcached running as part of Vellum nodes) so is not tied to any individual Sprout node.

Dime and Homer similarly only retain local state for pending requests - all long lived state is stored redundantly in the data store clusters provided by Vellum.

Cloud Security

SIP communications are divided into a trusted zone (for flows between Sprout nodes, Bono nodes and trusted application servers) and an untrusted zone (for message flows between Bono nodes and external clients or other systems). These zones use different ports allowing the trusted zone to be isolated using security groups and/or firewall rules, while standard SIP authentication mechanisms are used to protect the untrusted ports.

Other interfaces such as the XCAP and Homestead interfaces use a combination of locked down ports, standard authentication schemes and shared secret API keys for security.

Clearwater Tour

This document is a quick tour of the key features of Clearwater, from a subscriber's point of view.

Self-service account and line creation

In a browser, go to the Account Signup tab at `http://ellis.<domain>/signup.html` and supply details as requested:

- Use whatever name you like.
- Use a real email address, or the password reset function won't work.
- The signup code to enter in the final box should have been given to you by the deployment administrator.

- Hit "Sign up".

You are now shown a new account with a phone number (which may take a second or two to show up).

- The password for each number is only shown once - it is only stored encrypted in the system, for security, so cannot be retrieved by the system later. It can be reset by the user if they forget it (see button towards the right).

Try adding and deleting extra lines. Lines can have non-routable numbers for a purely on-net service, or PSTN numbers. Make sure you end up with just a PSTN number assigned to the account.

Address book: for this system there is a single address book which all numbers are by default added to - see the checkbox by the line.

Connecting an X-Lite client

Now register for the new PSTN line on your newly created account with your as yet unregistered X-Lite phone.

- Download X-Lite.
- Install and start X-Lite
- Navigate to Softphone / Account Settings
- Account tab:
  - Account name: Whatever you like.
  - User ID: "SIP Username" from the line you're adding
  - Domain: `<deployment domain>`, e.g., `ngv.example.com`
  - Password: "SIP Password" from the line you're adding
  - Display name: Whatever you like, or omit
  - Authorization name: `<SIP Username>@<domain>` e.g. `6505550611@ngv.example.com`
  - "Register with domain and receive calls" should be selected
  - Send outbound via: Domain
- Topology tab:
  - Select "Auto-detect firewall traversal method using ICE (recommended)"
  - Server address: `<domain>`
  - User name: `<SIP Username>@<domain>`
  - Password: "SIP Password" from the line
- Transport tab:
  - Signaling transport: TCP
- hit "OK"
- phone should register.

We have now registered for the new line.

Connecting an Android phone

See here for instructions.

Making calls

- Make a call using your new line to a PSTN number (eg your regular mobile phone number). Clearwater performs an ENUM lookup to decide how to route the call, and rings the number.
- Make a call using a PSTN number (eg your regular mobile phone) to your new line. Use the online global address book in Ellis to find the number to call.
- Make a call between X-Lite and the Android client, optionally using the address book.

We've gone from no user account, no line, and no service to a configured user and working line in five minutes. All of this works without any on-premise equipment, without dedicated servers in a data centre, without any admin intervention at all.

Supported dialling formats

A brief note on supported dialing formats:

- Dialling between devices on the Clearwater system. It is easiest to always dial the ten digit number, even when calling a PSTN number. Seven digit dialling is not supported, and dialing 1+10D is not supported. In theory dialling +1-10D if the destination is a Clearwater PSTN number should work, but some clients (eg. X-Lite) silently strip out the + before sending the INVITE, so only do this if you are absolutely sure your client supports it.
- Dialling out to the PSTN. You must dial as per the NANP. Calls to US numbers can use 10D, 1-10D or +1-10D format as appropriate (the latter only if your device supports +), and international calls must use the 011 prefix (+ followed by country code does not work). If you try to make a call using one of these number formats and it doesn't connect, then it almost certainly wouldn't connect if you were dialling the same number via the configured SIP trunk directly.

WebRTC support

See WebRTC support in Clearwater for how to use a browser instead of a SIP phone as a client.

VoLTE call services

Call barring

- From Ellis, press the Configure button on the X-Lite PSTN number, this will pop up the services management interface.
- Select the Barring tab and choose the bars you wish to apply to incoming/outgoing calls.
- Click OK then attempt to call X-Lite from the PSTN, see that the call is rejected without the called party being notified of anything.

Call diversion

- From Ellis, press the Configure button on the X-Lite PSTN number, this will pop up the services management interface.
- Select the Barring tab and turn off barring.
- Select the Redirect tab, enable call diversion and add a new rule to unconditionally divert calls to your Android client if using or your chosen call diversion destination if not.

- Click OK then call X-Lite from the PSTN and see that that number is not notified but the diverted-to number is, and can accept the call.

Other features

Clearwater also supports privacy, call hold, and call waiting.

Exploring Clearwater

After following the Install Instructions you will have a Clearwater deployment which is capable of handling on-net calls and has a number of MMTel services available through the built-in TAS.

Clearwater supports many optional extra features that you may chose to add to your deployment. Descriptions of each feature along with the process to enable it on your system can be found here.

Integrating Clearwater with existing services

An install of Clearwater will, by default, run with no external requirements and allow simple on-net calls between lines provisioned in Ellis. Clearwater also supports interoperating with various other SIP core devices to add more functionality and integrate with an existing SIP core.

- IBCF function
- Application Server
- External HSS Integration
- CDF Integration
- SIP RFCs supported

Call features

Clearwater has a number of call features provided by its built-in TAS, including:

- Call barring
- Call diversion
- Privacy

Clearwater's support for all IR.92 Supplementary Services is discussed in this document.

Scaling and Redundancy

Clearwater is designed from the ground up to scale horizontally across multiple instances to allow it to handle large subscriber counts and to support a high level of redundancy.

- Scaling Numbers
- The local redundancy story is described in the Clearwater Architecture page.
- Geographic redundancy

Operational support

To help you manage your deployment, Clearwater provides:

- Command-line/scriptable provisioning tools
- Support for separated management networks
- Deployment monitoring
- Backup
- A troubleshooting guide with instructions for viewing the database entries on Sprout and Homestead/Homer.

Support

There are 3 support mechanisms for Clearwater.

- Mailing list - This is for questions about Clearwater.
- Issues - These are for problems with Clearwater. If you're not sure whether some behavior is expected or not, it's best to ask on the mailing list first.
- Pull Requests - These allow you to contribute fixes and enhancements back to Clearwater.

Mailing List

The mailing list is managed at lists.projectclearwater.org. When posting, please try to

- provide a clear subject line
- provide as much diagnostic information as possible.

You're more likely to get your question answered quickly and correctly that way!

Issues

Clearwater uses github's Issues system to track problems. To access the Issues system, first determine which project the issue is with - ask in the forums if you're not sure. Then go to that component's repository page in github and click the Issues tab at the top.

Remember to do a search for existing issues covering your problem before raising a new one!

Pull Requests

Clearwater follows the "Fork & Pull" model of collaborative development, with changes being offered to the main Clearwater codebase via pull requests. If you're interested in contributing,

- thanks!
- see the github docs for how to create a pull request
- before creating a pull request, please make sure you've successfully
  - ensured the code matches our Ruby or C++ coding guidelines
  - compiled the code
  - run the unit tests for the component

  - run the live tests against your changes.

Troubleshooting and Recovery

This document describes how to troubleshoot some common problems, and associated recovery strategies.

General

- Clearwater components are monitored by monit and should be restarted by it if the component fails or hangs. You can check that components are running by issuing `monit status`. If components are not being monitored as expected, run `monit monitor <component>` to monitor it. To restart a component, we recommend using `service <component> stop` to stop the component, and allowing monit to automatically start the component again.
- The Clearwater diagnostics monitor detects crashes in native clearwater processes (Bono, Sprout and Homestead) and captures a diagnostics bundle containing a core file (among other useful information). A diagnostics bundle can also be created by running a command line script (`sudo cw-gather_diags`).
- By default each component logs to `/var/log/<service>/`, at log level 2 (which only includes errors and very high level events). To see more detailed logs you can enable debug logging; details for how to do this for each component are below. Note that if you want to run stress through your deployment, you should revert the log levels back to the default level.
- Changes to `shared_config` are detected each time `cw-upload_shared_config` is run (see Modifying Clearwater settings), and logged to `/var/log/syslog` on the node from which the configuration was changed.

Ellis

The most common problem from Ellis is it reporting "Failed to update server". This can happen for several reasons.

- If Ellis reports "Failed to update server" when allocating a new number (either after an explicit request or as part of creating a whole new account), check that ellis has free numbers to allocate. The create_numbers.py script is safe to re-run, to ensure that numbers have been allocated.
- Check the `/var/log/ellis/ellis-*.log` files. If these indicate that a timeout occurred communicating with Homer or Homestead-prov, check that the DNS entries for Homer and Homestead-prov exist and are configured correctly. If these are already correct, check Homer or Homestead-prov to see if they are behaving incorrectly.
- To turn on debug logging for Ellis write `log_level=5` to `/etc/clearwater/user_settings` (creating it if it doesn't exist already), then restart Ellis (`sudo service ellis stop` - it will be restarted by monit).

To examine Ellis' database, run `mysql` (as root), then type `use ellis;` to set the correct database. You can then issue standard SQL queries on the users and numbers tables, e.g. `SELECT * FROM users WHERE email = '<email address>'`.

Vellum

Problems on Vellum may include:

- Failing to read or write to the Cassandra database (only relevant if you deployment is using Homestead-Prov, Homer or Memento):

  - Check that Cassandra is running (`sudo monit status`). If not, check its `/var/log/cassandra/*.log` files.
  - Check that Cassandra is configured correctly. First access the command-line CQL interface by running `cqlsh`. There are 3 databases:
    * Type `use homestead_provisioning;` to set the provisioning database and then `describe tables;` - this should report `service_profiles`, `public`, `implicit_registration_sets` and `private`.
    * Type `use homestead_cache;` to set the cache database and then `describe tables;` - this should report `impi`, `impi_mapping` and `impu`.
    * Type `use homer;` to set the homer database and then `describe tables;` - this should report `simservs`.
    * If any of these are missing, recreate them by running
      · `/usr/share/clearwater/cassandra-schemas/homestead_cache.sh`
      · `/usr/share/clearwater/cassandra-schemas/homestead_provisioning.sh`
      · `/usr/share/clearwater/cassandra-schemas/homer.sh`
  - Check that Cassandra is clustered correctly (if running a multi-node system). `nodetool status` tells you which nodes are in the cluster, and how the keyspace is distributed among them.
  - To examine Vellum's database, run `cqlsh` and then type `use homer;`, `use homestead_provisioning;` or `use homestead_cache` to set the correct database. You can then issue CQL queries such as `SELECT * FROM impi WHERE private_id = '<private user ID>'`.

- Problems with the memcaced cluster:
  - It's a little clunky to examine this data but you can get some basic information out by running `. /etc/clearwater/config ; telnet $local_ip 11211` to connect to memcached, issuing `stats items`. This returns a list of entries of the form `STAT items:<slab ID>:...`. You can then query the keys in each of the slabs with `stats cachedump <slab ID> 0`.
  - Memcached logs to `/var/log/memcached.log`. It logs very little by default, but it is possible to make it more verbose by editing `/etc/memcached_11211.conf`, uncommenting the `-vv` line, and then restarting memcached

- Problems with the Chronos cluster.
  - Chronos logs to `/var/log/chronos/chronos*`.
  - Details of how to edit the Chronos configuration are here.

Sprout

The most common problem on Sprout is lack of communication with other nodes and processes, causing registration or calls to fail. Check that Vellum and Dime are reachable and responding.

To turn on debug logging for Sprout write `log_level=5` to `/etc/clearwater/user_settings` (creating it if it doesn't exist already), then restart Sprout (`sudo service sprout stop` - it will be restarted by monit).

If you see Sprout dying/restarting with no apparent cause in `/var/log/sprout/sprout*.txt`, check `/var/log/monit.log` and `/var/log/syslog` around that time - these can sometimes give clues as to the cause.

Bono

The most common problem on Bono is lack of communication with Sprout. Check that Sprout is reachable and responding.

To turn on debug logging for Bono write `log_level=5` to `/etc/clearwater/user_settings` (creating it if it doesn't exist already), then restart Bono (`sudo service bono stop` - it will be restarted by monit).

If you see Bono dying/restarting with no apparent cause in `/var/log/bono/bono*.txt`, check `/var/log/monit.log` and `/var/log/syslog` around that time - these can sometimes give clues as to the cause.

Dime

The most common problems on Dime are:

- Lack of communication with Vellum. Check that Vellum is reachable and responding.
- (If using Ralf) Lack of communication with a CCF. Check that your CCF is reachable and responding (if you don't have a CCF, you don't need Ralf).

To turn on debug logging for Ralf, Homestead or Homestead-prov write `log_level=5` to `/etc/clearwater/user_settings` (creating it if it doesn't exist already), then restart the service (`sudo service <ralf|homestead|homestead-prov> stop` - it will be restarted by monit).

If you see Ralf, Homestead or Homestead-prov dying/restarting with no apparent cause in `/var/log/<service>/<service>*.txt`, check `/var/log/monit.log` and `/var/log/syslog` around that time - these can sometimes give clues as to the cause.

Deployment Management

Clearwater comes with a system that automate clustering and configuration sharing. If you cannot scale your deployment up or down, or if configuration changes are not being applied, this system may not be working.

- The management system logs to `/var/log/clearwater-etcd`, `/var/log/clearwater-cluster-manager`, `/var/log/clearwater-config-manager` and `/var/log/clearwater-queue-manager`. To turn on debug logging write `log_level=5` to `/etc/clearwater/user_settings` (creating it if it doesn't exist already), then restart the etcd processes (`sudo service <clearwater-config-manager|clearwater-cluster-manager|clearwater-queue-manager> stop` - they will be restarted by monit)
- `cw-check_cluster_state` will display information about the state of the various data-store clusters used by Clearwater.
- `sudo cw-check_config_sync` will display whether the node has learned shared configuration.
- `sudo cw-check_restart_queue_state` will display whether there is new shared configuration that is being synched across the deployment, and which nodes are using the new shared configuration.
- The following commands can be useful for inspecting the state of the underlying etcd cluster used by the management system:

```
clearwater-etcdctl cluster-health
clearwater-etcdctl member list
```

Getting Help

If none of the above helped, please try the mailing list.

Installation Instructions

These pages will guide you through installing a Clearwater deployment from scratch.

Installation Methods

What are my choices?

1. All-in-one image, either using an AMI on Amazon EC2 or an OVF image on a private virtualization platform such as VMware or VirtualBox. This is the recommended method for trying Clearwater out.

   This installation method is very easy but offers no redundancy or scalability and relatively limited performance. As a result, it is great for familiarizing yourself with Clearwater before moving up to a larger-scale deployment using one of the following methods.

2. An automated install, using the Chef orchestration system. This is the recommended install for spinning up large scale deployments since it can handle creating an arbitrary sized deployment in a single command.

   This installation method does have its limitations though. It is currently only supported on Amazon's EC2 cloud and assumes that DNS is being handled by Amazon's Route53 and that Route53 controls the deployment's root domain. It also requires access to a running Chef server. Setting this up is non-trivial but only needs to be done once, after which multiple deployments can be spun up very easily.

3. A manual install, using Debian packages and hand configuring each machine. This is the recommended method if chef is not supported on your virtualization platform or your DNS is not provided by Amazon's Route53.

   This install can be performed on any collection of machines (at least 5 are needed) running Ubuntu 14.04 as it makes no assumptions about the environment in which the machines are running. On the other hand, it requires manually configuring every machine, firewalls and DNS entries, meaning it is not a good choice for spinning up a large-scale deployment.

4. Installing from source. If you are running on a non-Ubuntu-based OS, or need to test a code fix or enhancement you've made, you can also install Clearwater from source, building the code yourself. Per-component instructions are provided that cover the route from source code to running services. Familiarity with performing a manual install on Ubuntu will help with configuring your network correctly after the software is installed.

   If you install from source, especially on a non-Ubuntu OS, we'd love to hear about your experience, good or bad.

Installation Steps

The installation process for a full Clearwater deployment (i.e. not an all-in-one) can be described at a high level as follows:

- Sourcing enough machines to host the deployment (minimum is 6) and installing an OS on them all. Ubuntu 14.04 is the recommended and tested OS for hosting Clearwater.
- Preparing each machine to allow installation of the Clearwater software.
- Installing the Clearwater software onto the machines.
- Configuring firewalls to allow the various machines to talk to each other as required.

- Configuring DNS records to allow the machines to find each other and to expose the deployment to clients.

Detailed Instructions

Once you've decided on your install method, follow the appropriate link below.

- All-in-one Images
- Automated Install
- Manual Install
- For source installs, see the per-component instructions (Sprout/Bono, Ellis, Homer/Homestead).

If you hit problems during this process, see the Troubleshooting and Recovery steps.

If you want to deploy with IPv6 addresses, see the IPv6 notes.

Next Steps

Once you have installed your deployment, you'll want to test that it works.

- Making your first call
- Running the live test suite

All-in-one Images

While Clearwater is designed to be massively horizontally scalable, it is also possible to install all Clearwater components on a single node. This makes installation much simpler, and is useful for familiarizing yourself with Clearwater before moving up to a larger-scale deployment using one of the other installation methods.

This page describes the all-in-one images, their capabilities and restrictions and the installation options available.

Images

All-in-one images consist of

- Ubuntu 14.04, configured to use DHCP
- bono, sprout, homestead, homer and ellis
- Clearwater auto-configuration scripts.

On boot, the image retrieves its IP configuration over DHCP and the auto-configuration scripts then configure the bono, sprout, homestead, homer and ellis software to match.

The image is designed to run on a virtual machine with a single core, 2GB RAM and 8GB of disk space. On EC2, this is an t2.small.

Capabilities and Restrictions

Since the all-in-one images include all of bono, sprout, homestead, homer and ellis, they are capable of almost anything a full deployment is capable of.

The key restrictions of all-in-one images are

- hard-coded realm - the all-in-one image uses a hard-coded realm of `example.com` so your SIP URI might be `sip:6505551234@example.com` - by default, SIP uses this realm for routing but `example.com` won't resolve to your host so you need to configure an outbound proxy on all your SIP clients (more details later)
- performance - since all software runs on a single virtual machine, performance is significantly lower than even the smallest scale deployment
- scalability - there is no option to scale up and add more virtual machines to boost capacity - for this, you must create a normal deployment
- fault-tolerance - since everything runs on a single virtual machine, if that virtual machine fails, the service as a whole fails.

Installation Options

All-in-one images can be installed on EC2 or on your own virtualization platform, as long as it supports OVF (Open Virtualization Format).

- To install on EC2, follow the all-in-one EC2 AMI installation instructions.
- To install on your own virtualization platform, follow the all-in-one OVF installation instructions.

Manual Build

If your virtualization platform is not EC2 and doesn't support OVF, you may still be able to manually build an all-in-one node. To do so,

1. install Ubuntu 14.04 - 64bit server edition
2. find the `preseed/late_command` entry in the all-in-one image's install script - as of writing this is as follows, but please check the linked file for the latest version

```
d-i preseed/late_command string in-target bash -c '{ echo "#!/bin/bash" ; \
                                                     echo "set -e" ; \
                                                     echo "repo=... # filled in by make_
↪ovf.sh" ; \
                                                     echo "curl -L https://raw.
↪githubusercontent.com/Metaswitch/clearwater-infrastructure/master/scripts/
↪clearwater-aio-install.sh | sudo bash -s clearwater-auto-config-generic $repo" ;
↪ \
                                                     echo "python create_numbers.py --
↪start 6505550000 --count 1000" ; \
                                                     echo "rm /etc/rc2.d/S99zclearwater-
↪aio-first-boot" ; \
                                                     echo "poweroff" ; } > /etc/rc2.d/
↪S99zclearwater-aio-first-boot ; \
                                                   chmod a+x /etc/rc2.d/S99zclearwater-aio-
↪first-boot'
```

3. run the command (stripping the `d-i preseed/late_command string in-target` prefix, filling in the repo variable - the default is `http://repo.cw-ngv.com/stable`, and stripping the `\`) - this will create an `/etc/rc2.d/S99zclearwater-aio-first-boot` script
4. run the `/etc/rc2.d/S99zclearwater-aio-first-boot` script - this will install the all-in-one software and then shutdown (ready for an image to be taken)
5. restart the node.

Automated Install Instructions

These instructions will take you through preparing for an automated install of Clearwater using Chef. For a high level look at the install process, and a discussion of the various install methods, see Installation Instructions. The automated install is the suggested method for installing a large-scale deployment of Clearwater. It can also be used to install an all-in-one node.

The automated install is only supported for deployments running in Amazon's EC2 cloud, where DNS is being provided by Amazon's Route53 service. If your proposed deployment doesn't meet these requirements, you should use the Manual Install instructions instead.

The Install Process

The automated install system is made up of two phases:

- A series of steps that need to be performed once, before any deployment is created.
- A simpler set of instructions that will actually create the deployment.

Once the first phase has been completed, multiple deployments (of various sizes) can be created by repeating the second phase multiple times.

The first phase:

- Installing a Chef server
- This server will track the created Clearwater nodes and allow the client access to them.
- Configuring a Chef workstation
- This machine will be the one on which deployments will be defined and managed.

The second phase:

- Creating a deployment environment
- The automated install supports the existence and management of multiple deployments simultaneously, each deployment lives in an environment to keep them separate.
- Creating the deployment
- Actually provisioning the servers, installing the Clearwater software and configuring DNS.

Next steps

Once you've followed the instructions above, your Clearwater deployment is ready for use. Be aware that the newly created DNS entries may take some time to propagate fully through your network and until propagation is complete, your deployment may not function correctly. This propagation usually takes no more than 5 minutes.

To test the deployment, you can try making some real calls, or run the provided live test framework.

- Making your first call
- Running the live test framework

Manual Install Instructions

These instructions will take you through installing a minimal Clearwater system using the latest binary packages provided by the Clearwater project. For a high level look at the install process, and a discussion of alternative install methods, see Installation Instructions.

Prerequisites

Before starting this process you will need the following:

- Six machines running clean installs of Ubuntu 14.04 - 64bit server edition.
  - The software has been tested on Amazon EC2 `t2.small` instances (i.e. 1 vCPU, 2 GB RAM), so any machines at least as powerful as one of them will be sufficient.
  - Each machine will take on a separate role in the final deployment. The system requirements for each role are the same thus the allocation of roles to machines can be arbitrary.
  - The firewalls of these devices must be independently configurable. This may require some attention when commissioning the machines. For example, in Amazon's EC2, they should all be created in separate security groups.
  - On Amazon EC2, we've tested both within a VPC and without. If using a VPC, we recommend using the "VPC with a Single Public Subnet" model (in the "VPC Wizard") as this is simplest.
- A publicly accessible IP address of each of the above machines and a private IP address for each of them (these may be the same address depending on the machine environment). These will be referred to as `<publicIP>` and `<privateIP>` below. (If running on Amazon EC2 in a VPC, you must explicitly add this IP address by ticking the "Automatically assign a public IP address" checkbox on creation.)
- The FQDN of the machine, which resolves to the machine's public IP address (if the machine has no FQDN, you should instead use the public IP). Referred to as `<hostname>` below.
- SSH access to the above machines to a user authorised to use sudo. If your system does not come with such a user pre-configured, add a user with `sudo adduser <username>` and then authorize them to use sudo with `sudo adduser <username> sudo`.
- A DNS zone in which to install your deployment and the ability to configure records within that zone. This zone will be referred to as `<zone>` below.
- If you are not using the Project Clearwater provided Debian repository, you will need to know the URL (and, if applicable, the public GPG key) of your repository.

Configure the APT software sources

Configure each machine so that APT can use the Clearwater repository server.

Under sudo, create `/etc/apt/sources.list.d/clearwater.list` with the following contents:

```
deb http://repo.cw-ngv.com/stable binary/
```

Note: If you are not installing from the provided Clearwater Debian repository, replace the URL in this file to point to your Debian package repository.

Once this is created install the signing key used by the Clearwater server with:

```
curl -L http://repo.cw-ngv.com/repo_key | sudo apt-key add -
```

You should check the key fingerprint with:

```
sudo apt-key finger
```

The output should contain the following - check the fingerprint carefully.

```
pub   4096R/22B97904 2013-04-30
      Key fingerprint = 9213 4604 DE32 7DF7 FEB7  2026 111D BE47 22B9 7904
uid                  Project Clearwater Maintainers <maintainers@projectclearwater.
↪org>
sub   4096R/46EC5B7F 2013-04-30
```

Once the above steps have been performed, run the following to re-index your package manager:

```
sudo apt-get update
```

Determine Machine Roles

At this point, you should decide (if you haven't already) which of the six machines will take on which of the Clearwater roles.

The six roles are:

- Ellis
- Bono
- Sprout
- Homer
- Dime
- Vellum

Firewall configuration

We need to make sure the Clearwater nodes can all talk to each other. To do this, you will need to open up some ports in the firewalls in your network. The ports used by Clearwater are listed in Clearwater IP Port Usage. Configure all of these ports to be open to the appropriate hosts before continuing to the next step. If you are running on a platform that has multiple physical or virtual interfaces and the option to apply different firewall rules on each, make sure that you open these ports on the correct interfaces.

Create the per-node configuration.

On each machine create the file `/etc/clearwater/local_config` with the following contents.

```
local_ip=<privateIP>
public_ip=<publicIP>
public_hostname=<hostname>
etcd_cluster="<comma separated list of private IPs>"
```

Note that the `etcd_cluster` variable should be set to a comma separated list that contains the private IP address of the nodes you created above. For example if the nodes had addresses 10.0.0.1 to 10.0.0.6, `etcd_cluster` should be set to `"10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5,10.0.0.6"`

If you are creating a geographically redundant deployment, then:

- `etcd_cluster` should contain the IP addresses of nodes only in the local site
- You should set `local_site_name` in `/etc/clearwater/local_config`. The name you choose is arbitrary, but must be the same for every node in the site. This name will also be used in the `remote_site_names`, `sprout_registration_store`, `homestead_impu_store` and `ralf_session_store` configuration options set in shared config (desscribed below).
- If your deployment uses Homestead-Prov, Homer or Memento:
  - on the first Vellum node in the second site, you should set `remote_cassandra_seeds` to the IP address of a Vellum node in the first site.

Install Node-Specific Software

`ssh` onto each box in turn and follow the appropriate instructions below according to the role the node will take in the deployment:

Ellis

Install the Ellis package with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install ellis --yes
sudo DEBIAN_FRONTEND=noninteractive apt-get install clearwater-management --yes
```

Bono

Install the Bono and Restund packages with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install bono restund --yes
sudo DEBIAN_FRONTEND=noninteractive apt-get install clearwater-management --yes
```

Sprout

Install the Sprout package with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install sprout --yes
sudo DEBIAN_FRONTEND=noninteractive apt-get install clearwater-management --yes
```

If you want the Sprout nodes to include a Memento Application server, then install the Memento packages with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install memento-as memento-nginx --yes
```

Homer

Install the Homer packages with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install homer --yes
sudo DEBIAN_FRONTEND=noninteractive apt-get install clearwater-management --yes
```

Dime

Install the Dime package with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install dime clearwater-prov-tools --yes
sudo DEBIAN_FRONTEND=noninteractive apt-get install clearwater-management --yes
```

Vellum

Install the Vellum packages with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install vellum --yes
sudo DEBIAN_FRONTEND=noninteractive apt-get install clearwater-management --yes
```

If you included the Memento Application server on your Sprout nodes, then also install the required packages on Vellum with:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get install memento-cassandra --yes
```

SNMP statistics

Sprout, Bono, Vellum and Dime nodes expose statistics over SNMP. This function is not installed by default. If you want to enable it follow the instruction in our SNMP documentation.

Provide Shared Configuration

Log onto any node in the deployment and create the file `/etc/clearwater/shared_config` with the following contents. The `site_name` should match the value of `local_site_name` in `local_config`; if your deployment is not geographically redundant then you don't need to include it.

```
# Deployment definitions
home_domain=<zone>
sprout_hostname=sprout.<site_name>.<zone>
sprout_registration_store=vellum.<site_name>.<zone>
hs_hostname=hs.<site_name>.<zone>:8888
hs_provisioning_hostname=hs.<site_name>.<zone>:8889
homestead_impu_store=vellum.<zone>
ralf_hostname=ralf.<site_name>.<zone>:10888
ralf_session_store=vellum.<zone>
xdms_hostname=homer.<site_name>.<zone>:7888
chronos_hostname=vellum.<site_name>.<zone>
cassandra_hostname=vellum.<site_name>.<zone>

# Email server configuration
smtp_smarthost=<smtp server>
smtp_username=<username>
smtp_password=<password>
email_recovery_sender=clearwater@example.org

# Keys
signup_key=<secret>
turn_workaround=<secret>
ellis_api_key=<secret>
ellis_cookie_key=<secret>
```

If you wish to enable the optional external HSS lookups, add the following:

```
# HSS configuration
hss_hostname=<address of your HSS>
hss_port=3868
```

If you want to host multiple domains from the same Clearwater deployment, add the following (and configure DNS to route all domains to the same servers):

```
# Additional domains
additional_home_domains=<domain 1>,<domain 2>,<domain 3>...
```

If you want your Sprout nodes to include Gemini/Memento Application Servers add the following:

```
# Application Servers
gemini=<gemini port>
memento=<memento port>
memento_auth_store=vellum.<site_name>.<zone>
```

See the Chef instructions for more information on how to fill these in. The values marked `<secret>` **must** be set to secure values to protect your deployment from unauthorized access. To modify these settings after the deployment is created, follow these instructions.

If you are creating a geographically redundant deployment, some of the options require information about all sites to be specified. You need to set the `remote_site_names` configuration option to include the `local_site_name` of each site, replace the `sprout_registration_store`, `homestead_impu_store` and `ralf_session_store` with the values as described in Clearwater Configuration Options Reference, and set the `sprout_chronos_callback_uri` and `ralf_chronos_callback_uri` to deployment wide hostnames. For example, for sites named `siteA` and `siteB`:

```
remote_site_names=siteA,siteB
sprout_registration_store="siteA=vellum-siteA.<zone>,siteB=vellum-siteB.<zone>"
homestead_impu_store="siteA=vellum-siteA.<zone>,siteB=vellum-siteB.<zone>"
ralf_session_store="siteA=vellum-siteA.<zone>,siteB=vellum-siteB.<zone>"
sprout_chronos_callback_uri=sprout.<zone>
ralf_chronos_callback_uri=ralf.<zone>
```

Now run the following to upload the configuration to a shared database and propagate it around the cluster (see Modifying Clearwater settings for more details on this).

```
sudo cw-upload_shared_config
```

Provision Telephone Numbers in Ellis

Log onto you Ellis node and provision a pool of numbers in Ellis. The command given here will generate 1000 numbers starting at `sip:6505550000@<zone>`, meaning none of the generated numbers will be routable outside of the Clearwater deployment. For more details on creating numbers, see the create_numbers.py documentation.

```
sudo bash -c "export PATH=/usr/share/clearwater/ellis/env/bin:$PATH ;
              cd /usr/share/clearwater/ellis/src/metaswitch/ellis/tools/ ;
              python create_numbers.py --start 6505550000 --count 1000"
```

On success, you should see some output from python about importing eggs and then the following.

```
Created 1000 numbers, 0 already present in database
```

This command is idempotent, so it's safe to run it multiple times. If you've run it once before, you'll see the following instead.

```
Created 0 numbers, 1000 already present in database
```

DNS Records

Clearwater uses DNS records to allow each node to find the others it needs to talk to to carry calls. At this point, you should create the DNS entries for your deployment before continuing to the next step. Clearwater DNS Usage describes the entries that are required before Clearwater will be able to carry service.

Although not required, we also suggest that you configure individual DNS records for each of the machines in your deployment to allow easy access to them if needed.

Be aware that DNS record creation can take time to propagate, you can check whether your newly configured records have propagated successfully by running ''dig <record>'' on each Clearwater machine and checking that the correct IP address is returned.

If you are creating a geographically redundant deployment, you will also need to set up some DNS overrides. This allow a single hostname to be used across the deployment which can then be resolved to a site specific hostname at the point of use. This is necessary for Chronos and I-CSCF processing:

- Chronos: When a client sets a timer on Chronos, it provides a URI that Chronos can use to inform the client that the timer has popped. This URI should resolve to the clients in the same site as where the timer popped, but the timer could pop in any site.
- I-CSCF: The HSS stores the S-CSCF name. When the I-CSCF learns the S-CSCF name it wants to contact the S-CSCF in the local site, but the HSS will return the same S-CSCF name to the I-CSCFs in different sites.

Details for how to set up this DNS override are detailed here, and an example of the JSON file (for siteA) required for a GR deployment with two sites (siteA and siteB) is below:

```
{
  "hostnames": [
    {
      "name": "sprout.<zone>",
      "records": [{"rrtype": "CNAME", "target": "sprout.siteA.<zone>"}]
    },
    {
      "name": "scscf.sprout.<zone>",
      "records": [{"rrtype": "CNAME", "target": "scscf.sprout.siteA.<zone>"}]
    },
    {
      "name": "ralf.<zone>",
      "records": [{"rrtype": "CNAME", "target": "ralf.siteA.<zone>"}]
    }
  ]
}
```

Chronos configuration

Vellum nodes run the Chronos process, which is our distributed, redundant, reliable timer service (more information here). Chronos has three types of configuration; configuration that is local to an individual Chronos proces, configuration that covers how a Chronos process clusters with other Chronos processes in the same site, and configuration that

covers how a Chronos cluster connects to Chronos clusters in different sites. You don't typically need to set up the first two types of configuration, this is handled automatically. If you are creating a geographically redundant deployment, you do need to add the GR configuration for Chronos on each Vellum node - details of how to do this are here.

Where next?

Once you've reached this point, your Clearwater deployment is ready to handle calls. See the following pages for instructions on making your first call and running the supplied regression test suite.

- Making your first call
- Running the live test suite

Larger-Scale Deployments

If you're intending to spin up a larger-scale deployment containing more than one node of each types, it's recommended that you use the automated install process, as this makes scaling up and down very straight-forward. If for some reason you can't, you can add nodes to the deployment using the Elastic Scaling Instructions

Standalone IMS components and Application Servers

Our IMS components (I-CSCF, S-CSCF, ...) and application servers (Gemini, Memento, ...) can run on the same Sprout node, or they can be run as separate compoments/standalone application servers.

To install a standalone IMS component/application server, you need to: * Install a Sprout node (following the same process as installing a Sprout node above), but don't add the new node to the Sprout DNS cluster. * Enable/disable the sproutlets you want to run on this node - see here for more details on this. In particular, you should set the ports and the URIs of the sproutlets. * Once the node is fully installed, add it to the relevant DNS records.

I-CSCF configuration

The I-CSCF is responsible for sending requests to the correct S-CSCF. It queries the HSS, but if the HSS doesn't have a configured S-CSCF for the subscriber then it needs to select an S-CSCF itself. The I-CSCF defaults to selecting the Clearwater S-CSCF (as configured in `scscf_uri` in `/etc/clearwater/shared/config`).

You can configure what S-CSCFs are available to the I-CSCF by editing the `/etc/clearwater/s-cscf.json` file.

This file stores the configuration of each S-CSCF, their capabilities, and their relative weighting and priorities. The format of the file is as follows:

```
{
   "s-cscfs" : [
       {   "server" : "<S-CSCF URI>",
           "priority" : <priority>,
           "weight" : <weight>,
           "capabilities" : [<comma separated capabilities>]
       }
   ]
}
```

The S-CSCF capabilities are integers, and their meaning is defined by the operator. Capabilities will have different meanings between networks.

As an example, say you have one S-CSCF that supports billing, and one that doesn't. You can then say that capability 1 is the ability to provide billing, and your s-cscf.json file would look like:

```
{
   "s-cscfs" : [
       {   "server" : "sip:scscf1",
           "priority" : 0,
           "weight" : 100,
           "capabilities" : [1]
       },
       {   "server" : "sip:scscf2",
           "priority" : 0,
           "weight" : 100,
           "capabilities" : []
       }
   ]
}
```

Then when you configure a subscriber in the HSS, you can set up what capabilities they require in an S-CSCF. These will also be integers, and you should make sure this matches with how you've set up the s-cscf.json file. In this example, if you wanted your subscriber to be billed, you would configure the user data in the HSS to make it mandatory for your subscriber to have an S-CSCF that supports capability 1.

To change the I-CSCF configuration, edit this file on any Sprout node, then upload it to the shared configuration database by running `sudo cw-upload_scscf_json`.

Shared iFC configuration

If the configuration option `request_shared_ifcs` is set to 'Y', the S-CSCF must be configured with any Shared iFC sets that may be sent to it by the HSS.

You can configure Shared iFC sets on the S-CSCF by editing the `/etc/clearwater/shared_ifcs.xml` file.

This file stores the iFCs in each Shared iFC set. The format of the file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<SharedIFCsSets>
  <SharedIFCsSet>
    <SetID>  <set id>  </SetID>
    <InitialFilterCriteria>
      <iFC>
    </InitialFilterCriteria>
    <InitialFilterCriteria>
      <iFC>
    </InitialFilterCriteria>
  </SharedIFCsSet>
</SharedIFCsSets>
```

The `set id` is an integer, and each Shared iFCs set must have a unique `set id`.

The `iFC` is an iFC, in XML format.

There must be exactly one `SharedIFCsSets` element, which can contain multiple `SharedIFCsSet` elements (the minimum number of `SharedIFCsSet` elements is zero).

Each `SharedIFCsSet` element can contain multiple `InitialFilterCriteria` elements (the minimum number of `InitialFilterCriteria` elements is zero), and must contain exactly one unique `SetID` element.

To change the Shared iFC configuration, edit this file on any Sprout node, then upload it to the shared configuration database by running `sudo cw-upload_shared_ifcs_xml`.

To validate the Shared iFC configuration file before uploading it, run the command `cw-validate_shared_ifcs_xml <file_location>` on the Sprout node the file is present on.

To remove the Shared iFC configuration, run the command `sudo cw-remove_shared_ifcs_xml` on any Sprout node.

To view the Shared iFCs being used on any Sprout node, run the command `cw-display_shared_ifcs` on that Sprout node.

Fallback iFC configuration

If you wish to apply iFCs to any subscribers who have no iFCs triggered on a request (e.g. as a fallback to catch misconfigured subscribers), these iFCs must be configured on the S-CSCF, and the configuration option `apply_fallback_ifcs` set to 'Y'.

You can configure fallback iFCs on the S-CSCF by editing the `/etc/clearwater/fallback_ifcs.xml` file.

This file stores a list of fallback iFCs. The format of the file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<FallbackIFCsSet>
  <InitialFilterCriteria>
    <iFC>
  </InitialFilterCriteria>
</FallbackIFCsSet>
```

The `iFC` is an iFC, in XML format.

There must be exactly one `FallbackIFCsSet` element, which can can contain multiple `InitialFilterCriteria` elements (the minimum number of `InitialFilterCriteria` elements is zero).

To change the fallback iFC configuration, edit this file on any Sprout node, then upload it to the shared configuration database by running `sudo cw-upload_fallback_ifcs_xml`.

To validate the fallback iFC configuration file before uploading it, run the command `cw-validate_fallback_ifcs_xml <file_location>` on the Sprout node the file is present on.

To remove the fallback iFC configuration, run the command `sudo cw-remove_fallback_ifcs_xml` on any Sprout node.

To view the Fallback iFCs being used on any Sprout node, run the command `cw-display_fallback_ifcs` on that Sprout node.

All-in-one EC2 AMI Installation

This pages describes how to launch and run an all-in-one image in Amazon's EC2 environment.

Launch Process

Project Clearwater's all-in-one node is already available as a pre-built AMI, which can be found in the Community AMIs list on the US East region of EC2. Launching this follows exactly the same process as for other EC2 AMIs.

Before you launch the node, you will need an EC2 keypair, and a security group configured to provide access to the required ports.

To launch the node

- From the EC2 console, make sure you're in the US East region, then select "Instances", "Launch instance" and then "Classic Wizard"
- Select the "Community AMIs" tab, and search for "Clearwater"
- Press "Select" for the Clearwater all-in-one AMI. Take the most recent version unless you have a good reason not to.
- Choose the Instance Type you require (the node runs fine for basic functional testing on an t2.small)
- From the remaining pages of parameters, the only ones that need to be set are Name (give the node whatever convenient name you wish), keypair and security group.

On the last page, press "Launch", and wait for the node to be started by EC2.

Running and Using the Image

Once the node has launched, you can SSH to it using the keypair you supplied at launch time, and username `ubuntu`.

You can then try making your first call and running the live tests - for these you will need the signup key, which is `secret`. You will probably want to change this to a more secure value - see "Modifying Clearwater settings" for how to do this.

All-in-one OVF Installation

This pages describes how to install an all-in-one image on your own virtualization platform using OVF (Open Virtualization Format).

Supported Platforms

This process should work on any virtualization platform that supports OVFs using x86-64 CPUs, but has only been tested on

- VMware Player
- VirtualBox
- VMware ESXi.

The image uses DHCP to gets its IP configuration, so the virtualization platform must either serve DHCP natively or be connected to a network with a DHCP server.

If you install/run the OVF on another platform, please let us know how you get on on the mailing list so we can update this page.

Installation Process

To install the OVF, you must first download it from http://vm-images.cw-ngv.com/cw-aio.ova and save it to disk.

Then, you must import it into your virtualization platform. The process for this varies.

- On VMware Player, choose **File->Open a Virtual Machine** from the menu and select the cw-aio.ova file you downloaded. On the Import Virtual Machine dialog that appears, the defaults are normally fine, so you can just click **Import**.
- On VirtualBox, choose **File->Import Appliance...** from the menu. In the Appliance Import Wizard, click **Choose...**, select the cw-aio.ova file you downloaded and click **Next**. On the next tab, you can view the settings and then click **Import**.
- On VMware ESXi, using the VMware vSphere Client, choose **File->Deploy OVF Template...** from the menu. Select the cw-aio.ova file you downloaded and click through assigning it a suitable name, location and attached network (which must support DHCP) before finally clicking **Finish** to create the virtual machine.

Running and Using the Image

Once you've installed the virtual machine, you should start it in the usual way for your virtualization platform.

If you attach to the console, you should see an Ubuntu loading screen and then be dropped at a `cw-aio` login prompt. The username is `ubuntu` and the password is `cw-aio`. Note that the console is hard-coded to use a US keyboard, so if you have a different keyboard you might find that keys are remapped - in particular the - key.

The OVF provides 3 network services.

- SSH - username is `ubuntu` and password is `cw-aio`
- HTTP to ellis for subscriber management - sign-up code is `secret`. You will probably want to change this to a more secure value - see "Modifying Clearwater settings" for how to do this.
- SIP to bono for call signaling - credentials are provisioned through ellis.

How these network services are exposed can vary depending on the capabilities of the platform.

- VMware Player sets up a private network between your PC and the virtual machine and normally assigns the virtual machine IP address `192.168.28.128`. To access ellis, you'll need to point your browser at http://192.168.28.128. To register over SIP, you'll need to configure an outbound proxy of 192.168.28.128 port 5060.
- VirtualBox uses NAT on the local IP address, exposing SSH on port 8022, HTTP on port 8080 and SIP on port 8060. To access ellis, you'll need to point your browser at http://localhost:8080. To register over SIP, you'll need to configure an outbound proxy of localhost port 8060.
- VMware ESXi runs the host as normal on the network, so you can connect to it directly. To find out its IP address, log in over the console and type `hostname -I`. To access ellis, just point your browser at this IP address. To register over SIP, you'll need to configure an outbound proxy for this IP address.

Once you've successfully connected to ellis, try making your first call - just remember to configure the SIP outbound proxy as discussed above.

Installing a Chef server

This is the first step in preparing to install a Clearwater deployment using the automated install process. These instructions will guide you through installing a Chef server on an EC2 instance.

Prerequisites

- An Amazon EC2 account.

- A DNS root domain configured as a hosted zone with Route53 (Amazon's built-in DNS service, accessible from the EC2 console). This domain will be referred to as `<zone>` in this document.

Create the instance

Create a `t2.small` AWS EC2 instance running `Ubuntu Server 14.04.2 LTS` using the AWS web interface. The SSH keypair you provide here is referred to below as `<amazon_ssh_key>`. It is easiest if you use the same SSH keypair for all of your instances.

Configure its security group to allow access SSH, HTTP, and HTTPS access.

Configure a DNS entry for this machine, `chef-server.<zone>`. (The precise name isn't important, but we use this consistently in the documentation that follows.) It should have a non-aliased A record pointing at the public IP address of the instance as displayed in the EC2 console.

Once the instance is up and running and you can connect to it over SSH, you may continue to the next steps.

If you make a mistake, simply delete the instance permanently by selecting "Terminate" in the EC2 console, and start again. The terminated instance may take a few minutes to disappear from the console.

Install and configure the Chef server

The chef documentation explains how to install and configure the chef server. These instructions involve setting up a user and an organization.

- The user represents you as a user of chef. Pick whatever user name and password you like. We refer to these as `<chef-user-name>` and `<chef-user-password>`
- Organizations allow different groups to use the same chef server, but be isolated from one another. You can choose any organization name you like (e.g. "clearwater"). We refer to this as `<org-name>`

Follow steps 1-6 in the chef docs.

Once you have completed these steps, copy the `<chef-user-name>.pem` file off of the chef server - you will need it when installing a chef workstation.

Next steps

Once your server is installed, you can continue on to install a chef workstation.

Installing a Chef Client

These instructions cover commissioning a Chef client node on an EC2 server as part of the automated install process for Clearwater.

Prerequisites

- An Amazon EC2 account.
- A DNS root domain configured with Route53 (Amazon's built-in DNS service, accessible from the EC2 console. This domain will be referred to as `<zone>` in this document.
- You must have installed a Chef server and thus know the `<chef-user-name>` and `<chef-user-password>` for your server.

- A web-browser with which you can visit the Chef server Web UI.

Create the instance

Create a `t2.micro` AWS EC2 instance running `Ubuntu Server 14.04.2 LTS` using the AWS web interface. Configure its security group to allow access on port 22 (for SSH). The SSH keypair you provide here is referred to below as `<amazon_ssh_key>`. It is easiest if you use the same SSH keypair for all of your instances.

Configure a DNS entry for this machine, `chef workstation.<zone>`. (The precise name isn't important, but we use this consistently in the documentation that follows.) It should have a non-aliased A record pointing at the public IP address of the instance as displayed in the EC2 console.

Once the instance is up and running and you can connect to it over SSH, you may continue to the next steps.

If you make a mistake, simply delete the instance permanently by selecting "Terminate" in the EC2 console, and start again. The terminated instance may take a few minutes to disappear from the console.

Install Ruby 1.9.3

The Clearwater chef plugins use features from Ruby 1.9.3. To start run the following command.

```
curl -L https://get.rvm.io | bash -s stable
```

This may fail due to missing GPG signatures. If this happens it will suggest a command to run to resolve the problem (e.g. gpg –keyserver hkp://keys.gnupg.net –recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3'). Run the command suggested, then run the above command again, which should now succeed).

Next install the required ruby version.

```
source ~/.rvm/scripts/rvm
rvm autolibs enable
rvm install 1.9.3
rvm use 1.9.3
```

At this point, `ruby --version` should indicate that 1.9.3 is in use.

Installing the Clearwater Chef extensions

On the chef workstation machine, install git and dependent libraries.

```
sudo apt-get install git libxml2-dev libxslt1-dev
```

Clone the Clearwater Chef repository.

```
git clone -b stable --recursive git://github.com/Metaswitch/chef.git ~/chef
```

This will have created a `chef` folder in your home directory, navigate there now.

```
cd ~/chef
```

Finally install the Ruby libraries that are needed by our scripts.

```
bundle install
```

Creating a Chef user

You will need to configure yourself as a user on the chef server in order to use chef.

- If you are the person who created the chef server you wil already have added yourself as a user, and will know your username, organization name, and you will have a private key (<chef-user-name>, <org-name>, <chef-user-name>.pem respectively). These will be needed later.
- If you did not create the chef server, you will need to add an account for yourself. Log SSH on to the chef server and run the following commands, substituting in appropriate values for USER_NAME, FIRST_NAME, LAST_NAME, PASSWORD and ORG_NAME. We'll refer to the username as <chef-user-name> and the organization as <org-name>. This will create a <chef-user-name>.pem file in the current directory - save it for later.

```
sudo chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME EMAIL PASSWORD --
↪filename USER_NAME.pem
sudo chef-server-ctl org-user-add ORG_NAME USER_NAME --admin
```

Configure the chef workstation machine

Back on the chef workstation machine, create a .chef folder in your home directory.

```
mkdir ~/.chef
```

Copy the <chef-user-name>.pem file you saved off earlier to ~/.chef/<chef-user-name.pem>

Copy the validator key from the chef server to your client. You will need to either copy the Amazon SSH key to the client and use it, or copy the validator

```
scp -i <amazon_ssh_key>.pem ubuntu@chef-server.<zone>:<org-name>-validator.pem ~/.
↪chef/
```

or (on an intermediate box with the SSH key available)

```
scp -i <amazon_ssh_key>.pem ubuntu@chef-server.<zone>:<org-name>-validator.pem .
scp -i <amazon_ssh_key>.pem <org-name>-validator.pem ubuntu@chef workstation.<zone>:~/
↪.chef/
```

Configure knife using the built in auto-configuration tool.

```
knife configure
```

- Use the default value for the config location.
- The Chef server URL should be https://chef-server.<zone>/organizations/<org-name>
- The Chef client name should be <chef-user-name>
- The validation client name should be <org-name>-validator
- The validation key location should be ~/.chef/<org-name>-validator.pem.
- The chef repository path should be ~/chef/

Obtain AWS access keys

To allow the Clearwater extensions to create AWS instances or configure Route53 DNS entries, you will need to supply your AWS access key and secret access key. To find your AWS keys, you must be logged in as the main

AWS user, not an IAM user. Go to http://aws.amazon.com and click on `My Account/Console` then `Security Credentials`. From there, under the `Access Credentials` section of the page, click on the `Access Keys` tab to view your access key. The access key is referred to as `<accessKey>` below. To see your secret access key, just click on the `Show` link under `Secret Access Key`. The secret access key will be referred to as `<secretKey>` below.

Add deployment-specific configuration

Now add the following lines to the bottom of your `~/.chef/knife.rb` file, using the AWS deployment keys you obtained above.

```
# AWS deployment keys.
knife[:aws_access_key_id]     = "<accessKey>"
knife[:aws_secret_access_key] = "<secretKey>"
```

Test your settings

Test that knife is configured correctly

```
knife client list
```

This should return a list of clients and not raise any errors.

Next steps

At this point, the Chef server is up and running and ready to manage installs and the chef client is ready to create deployments. The next step is to create a deployment environment.

Creating a Chef deployment environment

These instructions make up part of the automated install process for Clearwater. They will take you through creating a deployment environment, which will be used to define the specific settings for a Clearwater deployment.

Prerequisites

- You must have installed a Chef workstation.
- You must have SSH access to the ubuntu user on the chef workstation machine.

Creating a keypair

You need to configure the AWS keypair used to access your Clearwater deployment. You may reuse the one you created when creating your Chef client and server; or you may create a new one (EC2 Dashboard > Network & Security > Key Pairs > Create Key Pair) for finer-grained access rights. The name you give your keypair will be referred to as `<keypair_name>` from this point.

Amazon will give you a `<keypair_name>`.pem file; copy that file to the `/home/ubuntu/.chef/` directory on your Chef client, and make it readable only to your user with `chmod 0700 /home/ubuntu/.chef ; chmod 0400 /home/ubuntu/.chef/<keypair_name>.pem`.

Creating the environment

Before creating an environment, choose a name (e.g. "clearwater") which will be referred to as `<name>` in the following steps. Now, on the chef workstation machine, create a file in `~/chef/environments/<name>.rb` with the following contents:

```
name "<name>"
description "Clearwater deployment - <name>"
cookbook_versions "clearwater" => "= 0.1.0"
override_attributes "clearwater" => {
  "root_domain" => "<zone>",
  "availability_zones" => ["us-east-1a", "us-east-1b"],
  "repo_servers" => ["http://repo.cw-ngv.com/stable"],
  "number_start" => "6505550000",
  "number_count" => 1000,
  "keypair" => "<keypair_name>",
  "keypair_dir" => "~/.chef/",
  "pstn_number_count" => 0,

  # Signup key. Anyone with this key can create accounts
  # on the deployment. Set to a secure value.
  "signup_key" => "secret",

  # TURN workaround password, used by faulty WebRTC clients.
  # Anyone with this password can use the deployment to send
  # arbitrary amounts of data. Set to a secure value.
  "turn_workaround" => "password",

  # Ellis API key. Used by internal scripts to
  # provision, update and delete user accounts without a password.
  # Set to a secure value.
  "ellis_api_key" => "secret",

  # Ellis cookie key. Used to prevent spoofing of Ellis cookies. Set
  # to a secure value.
  "ellis_cookie_key" => "secret",

  # SMTP credentials as supplied by your email provider.
  "smtp_server" => "localhost",
  "smtp_username" => "",
  "smtp_password" => "",

  # Sender to use for password recovery emails. For some
  # SMTP servers (e.g., Amazon SES) this email address
  # must be validated or email sending will fail.
  "email_sender" => "clearwater@example.com"
}
```

Fill in the values appropriate to your deployment using a text editor as directed.

- The value of `keypair` should *not* include the trailing .pem.
- The keys and passwords marked "Set to a secure value" above should be set to secure random values, to protect your deployment from unauthorised access. An easy way to generate a secure random key on a Linux system is as follows:

  ```
  head -c6 /dev/random | base64
  ```

The `signup_key` must be supplied by new users when they create an account on the system.

The `turn_workaround` must be supplied by certain WebRTC clients when using TURN. It controls access to media relay function.

The `ellis_api_key` and `ellis_cookie_key` are used internally.

- The SMTP credentials are required only for password recovery. If you leave them unchanged, this function will not work.
- For an all-in-one node the root-domain parameter is superfluous and will be ignored.

By default, your deployment will be created in the US East (North Virginia) region. However, if you want to deploy in another region, you must

- set the `region` property in the `override_attributes "clearwater"` block (e.g. to `us-west-2`)
- set the `availability_zones` property correspondingly (e.g. `["us-west-2a", "us-west-2b"]`)
- open `~/chef/plugins/knife/boxes.rb`, search for `@@default_image`, and comment in the EC2 AMI entry for the region you desire.

These fields override attributes defined and documented in the chef-base role.

If you want to use a different SIP registration period from the default (which is 5 minutes) add a line like `"reg_max_expires" => <timeout_in_secs>,` to the `override_attributes "clearwater"` block.

If you want to test geographic redundancy function, add `"num_gr_sites" => 2` to the environment file. This will create two logical 'sites' - they are not actually in different EC2 regions (cross-region security group rules make that difficult), but does allow you to see and test GR function.

To modify these settings after the deployment is created, follow these instructions.

Uploading the environment

The newly created environment needs to be uploaded to the Chef server before it can be used.

```
cd ~/chef
knife environment from file environments/<name>.rb
```

Next steps

At this point, your deployment environment is created and can be used to create a new deployment.

Creating a deployment with Chef

This is the final stage in creating a Clearwater deployment using the automated install process. Here we will actually create the deployment - commissioning servers and configuring DNS records.

Prerequisites

- You must have created the chef workstation machine and have SSH access to the ubuntu user on it.
- You must have created a deployment environment and know its name, `<name>`.

Upload Clearwater definitions to Chef server

The Chef server needs to be told the definitions for the various Clearwater node types. To do this, run

```
cd ~/chef
knife cookbook upload apt
knife cookbook upload chef-solo-search
knife cookbook upload clearwater
find roles/*.rb -exec knife role from file {} \;
```

You will need to do this step if the Clearwater definitions have never been uploaded to this Chef server before, or if the cookbooks or roles have changed since the last upload.

Note that cookbooks are versioned, but roles aren't, so uploading a changed role will affect other people deploying Clearwater from the same Chef server.

Creating a Deployment

You now have two options - you can create an All-in-One node, where all the Clearwater components are run on a single machine instance, or a larger deployment which can potentially have numerous instances of each component.

Creating an All-in-One ("AIO") node

To create a single machine instance running all the Clearwater components, run the following on the chef workstation machine.

```
cd ~/chef
knife box create cw_aio -E <name>
```

Our all-in-one nodes are created with the signup key 'secret', not the value configured in `knife.rb` (which is used for non-all-in-one nodes).

Optional arguments

The following modifier is available.

- `--index <value>` - Name the new node `<name>-cw_aio-<value>` to permit distinguishing it from others.

Creating a larger deployment

To kick off construction of the deployment, run the following on the chef workstation machine.

```
cd ~/chef
knife deployment resize -E <name> -V
```

Follow the on-screen prompts.

This will:

- Commission AWS instances
- Install the Clearwater software
- Configure security groups

- Configure DNS
- Start the Clearwater services

Optional arguments

The following modifiers are available to set the scale of your deployment.

- `--bono-count NUM` - Create `NUM` Bono nodes (default is 1)
- `--sprout-count NUM` - Create `NUM` Sprout nodes (default is 1)
- `--homer-count NUM` - Create `NUM` Homer nodes (default is 1)
- `--dime-count NUM` - Create `NUM` Dime nodes (default is 1)
- `--vellum-count NUM` - Create `NUM` Vellum nodes (default is 1)
- `--subscribers NUM` - Auto-scale the deployment to handle `NUM` subscribers.
- Due to a known limitation of the install process, Ellis will allocate 1000 numbers regardless of this value.
- To bulk provision subscribers (without using Ellis), follow these instructions

More detailed documentation on the available Chef commands is available here.

Next steps

Now your deployment is installed and ready to use, you'll want to test it.

- Making your first call
- Running the live tests

Making calls through Clearwater

These instructions will take you through the process of making a call on a Clearwater deployment.

Prerequisites

- You've installed Clearwater
- You have access to two SIP clients.
- If you have installed Clearwater on VirtualBox using the All-In-One image you must use Zoiper as one of your clients. For the other client (or for other install modes) you may use any standard SIP client, we've tested with the following:
  - X-Lite
  - Bria
  - Jitsi
  - Blink
  - Stock Android SIP client
  - Zoiper Android/iOS SIP client

  - Media5-fone iOS SIP client (on iPhone 4 and 4S)
- You have access to a web-browser. We've tested with:
- Google Chrome

Work out your base domain

If you installed Clearwater manually, your base DNS name will simply be `<zone>`. If you installed using the automated install process, your base DNS name will be `<name>.<zone>`. If you installed an All-in-One node, your base name will be `example.com`.

For the rest of these instructions, the base DNS name will be referred to as `<domain>`.

Work out your All-in-One node's identity

This step is only required if you installed an All-in-One node, either from an AMI or an OVF. If you installed manually or using the automated install process, just move on to the next step.

If you installed an All-in-One node from an Amazon AMI, you need the public DNS name that EC2 has assigned to your node. This will look something like `ec2-12-34-56-78.compute-1.amazonaws.com` and can be found on the EC2 Dashboard on the "instances" panel.

If you installed an All-in-One node from an OVF image in VMPlayer or VMWare, you need the IP address that was assigned to the node via DHCP. You can find this out by logging into the node's console and typing `hostname -I`.

If you installed an All-in-One node from an OVF in VirtualBox, you simply need `localhost`.

For the rest of these instructions, the All-in-One node's identity will be referred to as `<aio-identity>`.

Work out your Ellis URL

If you installed Clearwater manually or using the automated install process, your Ellis URL will simply be `http://ellis.<domain>`. If you installed an All-in-One node, your Ellis URL will be `http://<aio-identity>`.

Create a number for your client

In your browser, navigate to the Ellis URL you worked out above.

Sign up as a new user, using the signup code you set as `signup_key` when configuring your deployment.

Ellis will automatically allocate you a new number and display its password to you. Remember this password as it will only be displayed once. From now on, we will use `<username>` to refer to the SIP username (e.g. `6505551234`) and `<password>` to refer to the password.

Configure your client

Client configuration methods vary by client, but the following information should be sufficient to allow your client to register with Clearwater.

- SIP Username: `<username>`
- SIP Password: `<password>`
- SIP Domain: `<domain>`

- Authorization Name: `<username>@<domain>`
- Transport: `TCP`
- STUN/TURN/ICE:
- Enabled: `true`
- Server: `<domain>` (or `<aio-identity>` on an All-in-One node)
- Username: `<username>@<domain>`
- Password: `<password>`
- Use rport: `true` (this is required if your SIP client is behind a NAT when contacting your deployment).

Extra configuration to use an All-in-One node

If you are using an All-in-One node, you will also need to configure an outbound proxy at your client.

- Outbound Proxy address: `<aio-identity>`
- Port: 5060 (or 8060 if installed in VirtualBox)

Once these settings have been applied, your client will register with Clearwater. Note that X-Lite may need to be restarted before it will set up a STUN connection.

Configure a second number and client

Create a new number in Ellis, either by creating a new Ellis user, or by clicking the `Add Number` button in the Ellis UI to add one for the existing user.

Configure a second SIP client with the new number's credentials as above.

Make the call

From one client (Zoiper if running an All-in-One node in VirtualBox), dial the `<username>` of the other client to make the call. Answer the call and check you have two-way media.

Next steps

Now that you've got a basic call working, check that all the features of your deployment are working by running the live tests or explore Clearwater to see what else Clearwater offers.

Using the Android SIP Client with Clearwater

These instructions detail how to enable the stock Android SIP client. The instructions vary depending on your handset and Android model, in particular the Nexus 6 running 5.1.1 has different menus to the other known supported devices listed. If you get them working on another device, please add to the list. Equally, if they do not work, please add your device to the unsupported list.

Instructions

1. Ensure that you have a data connection (either through 3G or WiFi).
2. Launch your standard dialer (often called Phone or Dialer).
3. Bring up the menu. Depending on your phone, this may be via a physical Menu button, or via an icon (consisting of three dots in a vertical line in either the top right or bottom right).
4. From the menu, navigate to the internet calling accounts menu.
   - On many devices this is located in **Settings** (or **Call Settings**) > **Internet Call Settings** > **Accounts**.
   - On the Nexus 6 running 5.1.1 this is located in **Settings** > **Calls** > **Calling accounts** > **Internet Call Settings** > **Internet calling (SIP) accounts**.
5. Press **Add account** and enter its configuration.
   - Set **Username**, **Password** and **Server** to their usual values.
   - Open the **Optional settings** panel to see and set the following options.
   - Set **Transport type** to TCP.
   - Set **Authentication username** to "sip:<*username*>@<*server*>", substituting (note: once we switch to storing user digest by private id, this will change)
   - Some SIP clients do not have an **Authentication username** field. To make calls through Clearwater using these clients you will have to configure an external HSS to allow one subscriber to have two private identities. This will allow the subscriber to have one private identity of the form **''1234@example.com''** (the IMS standard form), and one private identity **''1234''** (which the phone will default to in the absence of a Authentication username).
6. Once done, choose **Save**.
7. Enable making internet calls.
   - On many devices this is located in the **Call** menu (which can be accessed from the dialler by selecting **Settings** as above). Once here under **Internet Call Settings** select **Use Internet Calling** and set this to **Ask for each call**.
   - On a Nexus 6 running 5.1.1 this is located in the **Calling accounts** menu. From here select **Use internet calling** and set this to **Only for Internet Calls**.
8. Enable receiving internet calls (note this significantly reduces your battery life).
   - On many devices this is located in **Settings** > **Internet Call Settings** > **Accounts** once here select **Receive Incoming Calls**.
   - On the Nexus 6 running 5.1.1 this is located in **Settings** > **Calls** > **Calling accounts** once here select **Receive Incoming Calls**.

Adding a SIP contact to your phone book

- For many devices:
  - navigate to the **People** App. Select **+** in the top right to add a new contact to the phone book.
  - Under **Add more information** select **Add another field**. In this menu choose **Internet call** and enter your contact's SIP address.
- For the Nexus 6 running 5.1.1:
  - navigate to the **Contacts** App. Select **+** in the bottom right to add a new contact to the phone book.

  - Under the **Sip** field enter your contact's SIP address.

Making a call

- Make a call from the standard dialler to another Clearwater line, e.g. 6505551234 and hit the Call button.
- You will be prompted whether to use internet calling. Select internet calling and your call should connect.

Known supported devices

- Samsung Galaxy Nexus - 4.2.2
- Orange San Francisco II - 2.3.5
- HTC One M9 - 5.1
- Samsung Galaxy S4 - 4.3
- Nexus 6 - 5.1.1

Using Zoiper with Clearwater

These instructions detail how to configure the Zoiper Android/iOS SIP client to work against a Clearwater system.

Instructions

1. Download the application from the Play Store or iTunes.
2. Once installed, go to Config -> Accounts -> Add account -> SIP
3. Fill in the following details:
   - Account name: Clearwater
   - Host: the root domain of your Clearwater deployment
   - Username: your username, e.g. 6505551234
   - Password: password for this account
   - Authentication user: <username>@<server> e.g. 6505551234@my-clearwater.com
4. Hit Save
5. If your account was successfully enabled you should see a green tick notification
6. Go back to the main Config menu and select Codecs
7. Unselect everything except uLaw and aLaw.
8. Hit Save
9. You are now ready to make calls.

Known supported devices

- Samsung Galaxy S3
- Samsung Galaxy S2
- Samsung Galaxy S4
- Nexus One
- Huawei G510
- Huawei G610
- Qmobile A8
- HTC 1X
- iPhone 4
- iPhone 4S

Known unsupported devices

- Galaxy Nexus (no media)

Configuring Geographic redundancy

This article describes

- How to install a geographically redundant deployment
- How to add a site to your deployment
- How to remove a site from your deployment

More information about Clearwater's geographic redundancy support is available here, and how to recover a deployment after site failure is described here.

Installing a geographically redundant deployment

The process for setting up a geographically-redundant deployment is as follows.

1. Create your first site, following the instructions here - make sure you add the extra configuration options/Chronos configuration for GR and set up your DNS records following the extra GR instructions. At this point, you have a working non-GR deployment.
2. Then add your second site, again following the instructions here, and again making sure you add the extra configuration options/Chronos configuration/DNS records for GR.
3. And that's it! Your two sites are ready to handle calls and can replicate data between themselves.

You can also use Chef to try GR function, by setting `"num_gr_sites" => 2` in the environment, as described in the automated install docs.

Adding a site to a non-GR deployment

Adding a site to a non-GR deployment follows the same basic process as described above. At various points in the steps below the instructions as for the site name; this is either the value of `local_site_name` in `/etc/clearwater/local_config` if it was set when the site was first created, or it's `site1`.

1. Add your second site, following the instructions here - make sure you add the extra configuration options/Chronos configuration for GR and set up your DNS records following the extra GR instructions. At this point, your second site is clustered with your first site, but no traffic is being sent to it.
2. Now you need to update the shared configuration on your first site so that it will communicate with your second site
   - Update the shared configuration on your first site to use the GR options - follow the GR parts of setting up shared config here.
   - Update the Chronos configuration on your Vellum nodes on your first site to add the GR configuration file - instructions here.
   - If you are using any of Homestead-Prov, Homer or Memento:
     - Update Cassandra's strategy by running `cw-update_cassandra_strategy` on any Vellum node in your entire deployment.
   - At this point, your first and second sites are replicating data between themselves, but no external traffic is going to your second site.
3. Change DNS so that your external nodes (e.g. the HSS, the P-CSCF) will send traffic to your new site. Now you have a GR deployment.

To revert this process, simply remove the new site, following the instructions below.

Removing a site from a GR deployment

The process for removing a site is as follows.

1. Change DNS so that no external nodes (e.g. the HSS, the P-CSCF) send any traffic to the site you're removing, and wait for the DNS TTL to pass. At this point your two sites are still replicating data between themselves, but the site you're removing isn't getting any external traffic.
2. Now you need to update the shared configuration on the site you're keeping so that it no longer replicates data to the site you're removing.
   - Update the Chronos configuration on your Vellum nodes on the remaining site to remove the GR configuration file - instructions here.
   - Update the shared configuration on your remaining site to remove the GR options - e.g. revert the GR parts of setting up shared config here.
   - At this point, your remaining site is handling all the external traffic, and isn't replicating any data to the site you're removing.
3. Now you can remove the site. Firstly, quiesce the main processes on each node in the site you're removing.
   - Bono - `monit unmonitor -g bono && sudo service bono quiesce` (note, this will take as long as your reregistration period)
   - Sprout or Memento- `monit unmonitor -g sprout && sudo service sprout quiesce`
   - Dime - `monit stop -g homestead && monit stop -g homestead-prov && monit stop -g ralf`

- Homer - `monit stop -g homer`
- Ellis - `monit stop -g ellis`
- Vellum - `sudo monit unmonitor -g clearwater_cluster_manager && sudo service clearwater-cluster-manager decommission`

4. Once this is complete, you can permanently delete the nodes.

To revert this process, simply add the site back, following the instructions above

Upgrading a Clearwater Deployment

This article explains how to upgrade a Project Clearwater deployment.

Quick Upgrade

The quickest way to upgrade is to simply upgrade all your nodes in any order. There is no need to wait for one node to finish upgrading before moving onto the next.

This is recommended if you:

- Do not have a fault-tolerant deployment (e.g. the All-in-One image, or a deployment with one node of each type) OR
- Do not need to provide uninterrupted service to users of your system.

Note that during a quick upgrade your users may be unable to register, or make and receive calls. If you do have a fault-tolerant deployment and need uninterrupted service, see "Seamless Upgrade" below.

Manual Install

If you installed your system using the Manual Install Instructions, run `sudo apt-get update -o Dir::Etc::sourcelist="sources.list.d/clearwater.list" -o Dir::Etc::sourceparts="-" -o APT::Get::List-Cleanup="0" && sudo apt-get install clearwater-infrastructure && sudo clearwater-upgrade` on each node.

Chef Install

If you installed your deployment with chef:

- Follow the instructions to update the Chef server
- Run `sudo apt-get update -o Dir::Etc::sourcelist="sources.list.d/clearwater.list" -o Dir::Etc::sourceparts="-" -o APT::Get::List-Cleanup="0" && sudo apt-get install clearwater-infrastructure && sudo chef-client && sudo clearwater-upgrade` on each node.

Seamless Upgrade

If your deployment contains at least two nodes of each type (excluding Ellis) it is possible to perform a *seamless upgrade* that does not result in any loss of service.

To achieve this the nodes must be upgraded one at a time and in a specific order: first all Vellum nodes, then Dime nodes (if present), Homer, Sprout, Memento (if present), Gemini (if present), Bono, and finally Ellis.

For example if your deployment has two Dimes, two Vellums, two Homers, two Sprouts, two Bonos, and one Ellis, you should upgrade them in the following order:

- Vellum-1
- Vellum-2
- Dime-1
- Dime-1
- Homer-1
- Homer-2
- Sprout-1
- Sprout-2
- Bono-1
- Bono-2
- Ellis

Manual Install

If you installed your system using the Manual Install Instructions run `sudo apt-get update -o Dir::Etc::sourcelist="sources.list.d/clearwater.list" -o Dir::Etc::sourceparts="-" -o APT::Get::List-Cleanup="0" && sudo apt-get install clearwater-infrastructure && sudo clearwater-upgrade` on each node in the order described above.

Chef Install

If you installed your deployment with chef:

- Follow the instructions to update the Chef server
- Run `sudo apt-get update -o Dir::Etc::sourcelist="sources.list.d/clearwater.list" -o Dir::Etc::sourceparts="-" -o APT::Get::List-Cleanup="0" && sudo apt-get install clearwater-infrastructure && sudo chef-client && sudo clearwater-upgrade` on each node in the order described above.

Modifying Clearwater Settings

This page discusses how to change settings on a Clearwater system. Most settings can be changed on an existing deployment (e.g. security keys and details of external servers), but some are so integral to the system (e.g. the SIP home domain) that the best way to change it is to recreate the Clearwater deployment entirely.

Modifying Settings in /etc/clearwater/shared_config

This will have a service impact of up to five minutes.

Settings in `/etc/clearwater/shared_config` can be safely changed without entirely recreating the system. The one exception to this is the `home_domain`; if you want to change this go to the "Starting from scratch" section instead.

To change one of these settings:

- Edit `/etc/clearwater/shared_config` on *one* node in each site and change to the new value.
- Run `sudo cw-upload_shared_config` to upload the new config to etcd. The changes to the shared configuration are logged to `/var/log/syslog` and to the console. Each node in the site picks up the changed shared configuration (using Clearwater's automatic configuration sharing functionality) and safely restarts itself to use it.
- You can check which nodes are using the new shared config by running `cw-check_restart_queue_state`. If this command shows that there's been an error (i.e. a node wasn't able to restart after picking up the new config), simply fix the `/etc/clearwater/shared_config` and run the `sudo cw-upload_shared_config` script again.

Modifying Sprout JSON Configuration

This configuration can be freely modified without impacting service.

Some of the more complex sprout-specific configuration is stored in JSON files

- `/etc/clearwater/s-cscf.json` - contains information to allow the Sprout I-CSCF to select an appropriate S-CSCF to handle some requests.
- `/etc/clearwater/bgcf.json` - contains routing rules for the Sprout BGCF.
- `/etc/clearwater/enum.json` - contains ENUM rules when using file-based ENUM instead of an external ENUM server.

To change one of these files:

- Edit the file on *one* of your sprout nodes in each site.
- Run one of `sudo cw-upload_{scscf|bgcf|enum}_json` depending on which file you modified.
- The change will be automatically propagated around the site (by Clearwater's automatic configuration sharing functionality) and will start being used.

Modifying Sprout XML Configuration

This configuration can be freely modified without impacting service.

Some of the more complex sprout-specific configuration is stored in XML files.

- `/etc/clearwater/shared_ifcs.xml` - contains information about all Shared iFC sets for the Sprout S-CSCF to use.
- `/etc/clearwater/fallback_ifcs.xml` - contains information about all the fallback iFCs for the Sprout S-CSCF to apply.

To change one of these files:

- Edit the file on *one* of your sprout nodes in each site.
- Run one of `sudo cw-upload_{shared|fallback}_ifcs_xml` depending on which file you modified.
- The change will be automatically propogated around the site (by Clearwater's automatic configuration sharing functionality) and will start being used.

Modifying DNS Config

This configuration can be freely modified without impacting service

It's possible to add static overrides to DNS to a clearwater node. This is done by adding entries to `/etc/clearwater/dns.json`. Currently, only CNAME records are supported and this is only used by Chronos and the I-CSCF.

To change this file:

- Edit the file on one of your nodes in each site.
- Run `sudo cw-upload_dns_json`.
- The change will be automatically propagated around the site (using Clearwater's automatic configuration sharing functionality) and will start being used.

Starting from scratch

This will have a service impact of up to half an hour.

If other settings (such as the Clearwater home domain) are being changed, we recommend that users delete their old deployment and create a new one from scratch, either with Chef or manually. This ensures that the new settings are consistently applied.

Clearwater Configuration Options Reference

This document describes all the Clearwater configuration options that can be set in `/etc/clearwater/shared_config`, `/etc/clearwater/local_config` or `/etc/clearwater/user_settings`.

At a high level, these files contain the following types of configuration options: * `shared_config` - This file holds settings that are common across the entire deployment. This file should be identical on all nodes (and any changes can be easily synchronised across the deployment as described in this process). * `local_config` - This file holds settings that are specific to a single node and are not applicable to any other nodes in the deployment. They are entered early on in the node's life and are not typically changed. * `user_settings` - This file holds settings that may vary between systems in the same deployment, such as log level (which may be increased on certain nodes to track down specific issues) and performance settings (which may vary if some nodes in your deployment are more powerful than others)

Modifying Configuration

You should follow this process when changing settings in "Shared Config". For settings in the "Local config" or "User settings" you should:

- Modify the configuration file
- Run `sudo service clearwater-infrastructure restart` to regenerate any dependent configuration files
- Restart the relevant Clearwater service(s) using the following commands as appropriate for the node.
  - Sprout - `sudo service sprout quiesce`
  - Bono - `sudo service bono quiesce`
  - Dime - `sudo service homestead stop && sudo service homestead-prov stop && sudo service ralf stop`

  - Homer - `sudo service homer stop`
  - Ellis - `sudo service ellis stop`
  - Memento - `sudo service memento stop`
  - Vellum - `sudo service astaire stop`

Local Config

This section describes settings that are specific to a single node and are not applicable to any other nodes in the deployment. They are entered early on in the node's life and are not normally changed. These options should be set in `/etc/clearwater/local_config`. Once this file has been created it is highly recommended that you do not change it unless instructed to do so. If you find yourself needing to change these settings, you should destroy and recreate then node instead.

- `local_ip` - this should be set to an IP address which is configured on an interface on this system, and can communicate on an internal network with other Clearwater nodes and IMS core components like the HSS.
- `public_ip` - this should be set to an IP address accessible to external clients (SIP UEs for Bono, web browsers for Ellis). It does not need to be configured on a local interface on the system - for example, in a cloud environment which puts instances behind a NAT.
- `public_hostname` - this should be set to a hostname which resolves to `public_ip`, and will communicate with only this node (i.e. not be round-robined to other nodes). It can be set to `public_ip` if necessary.
- `node_idx` - an index number used to distinguish this node from others of the same type in the cluster (for example, sprout-1 and sprout-2). Optional.
- `etcd_cluster` - this is either blank or a comma separated list of IP addresses, for example `etcd_cluster=10.0.0.1,10.0.0.2`. The setting depends on the node's role:
  - If this node is an etcd master, then it should be set in one of two ways:
    * If the node is forming a new etcd cluster, it should contain the IP addresses of all the nodes that are forming the new cluster as etcd masters (including this node).
    * If the node is joining an existing etcd cluster, it should contain the IP addresses of all the nodes that are currently etcd masters in the cluster.
  - If this node is an etcd proxy, it should be left blank
- `etcd_proxy` - this is either blank or a comma separated list of IP addresses, for example `etcd_proxy=10.0.0.1,10.0.0.2`. The setting depends on the node's role:
  - If this node is an etcd master, this should be left blank
  - If this node is an etcd proxy, it should contain the IP addresses of all the nodes that are currently etcd masters in the cluster.
- `etcd_cluster_key` - this is the name of the etcd datastore clusters that this node should join. It defaults to the function of the node (e.g. a Vellum node defaults to using 'vellum' as its etcd datastore cluster name when it joins the Cassandra cluster). This must be set explicitly on nodes that colocate function.
- `remote_cassandra_seeds` - this is used to connect the Cassandra cluster in your second site to the Cassandra cluster in your first site; this is only necessary in a geographically redundant deployment which is using at least one of Homestead-Prov, Homer or Memento. It should be set to an IP address of a Vellum node in your first site, and it should only be set on the first Vellum node in your second site.
- `scscf_node_uri` - this can be optionally set, and only applies to nodes running an S-CSCF. If it is configured, it almost certainly needs configuring on each S-CSCF node in the deployment.

If set, this is used by the node to advertise the URI to which requests to this node should be routed. It should be formatted as a SIP URI.

This will need to be set if the local IP address of the node is not routable by all the application servers that the S-CSCF may invoke. In this case, it should be configured to contain an IP address or host which is routable by all of the application servers – e.g. by using a domain and port on which the sprout can be addressed - `scscf_node_uri=sip:sprout-4.example.net:5054`.

The result will be included in the Route header on SIP messages sent to application servers invoked during a call.

If it is not set, the URI that this S-CSCF node will advertise itself as will be `sip:<local_ip>:<scscf_port>` where `<local_ip>` is documented above, and `<scscf_port>` is the port on which the S-CSCF is running, which is 5054 by default.

Shared Config

This section describes settings that are common across the entire deployment.

Core options

This section describes options for the basic configuration of a Clearwater deployment - such as the hostnames of the six node types and external services such as email servers or the Home Subscriber Server. These options should be set in the `/etc/clearwater/shared_config` file (in the format `name=value`, e.g. `home_domain=example.com`).

- `home_domain` - this is the main SIP domain of the deployment, and determines which SIP URIs Clearwater will treat as local. It will usually be a hostname resolving to all the P-CSCFs (e.g. the Bono nodes). Other domains can be specified through additional_home_domains, but Clearwater will treat this one as the default (for example, when handling `tel:` URIs).
- `sprout_hostname` - a hostname that resolves by DNS round-robin to the signaling interface of all Sprout nodes in the cluster.
- `sprout_hostname_mgmt` - a hostname that resolves by DNS round-robin to the management interface of all Sprout nodes in the cluster. Should include the HTTP port (always 9886). For details on the HTTP API exposed on this interface, see https://github.com/Metaswitch/sprout/blob/dev/docs/ManagementHttpAPI.md.
- `hs_hostname` - a hostname that resolves by DNS round-robin to the signaling interface of all Dime nodes in the cluster. Should include the HTTP port (always 8888). This is also used (without the port) as the Origin-Realm of the Diameter messages the homestead process on Dime sends.
- `hs_hostname_mgmt` - a hostname that resolves by DNS round-robin to the management interface of all Dime nodes in the cluster. Should include the HTTP port (always 8886). For details on the HTTP API exposed on this interface, see https://github.com/Metaswitch/homestead/blob/dev/docs/ManagementHttpAPI.md.
- `hs_provisioning_hostname` - a hostname that resolves by DNS round-robin to the management interface of all Dime nodes in the cluster. Should include the HTTP provisioning port (usually 8889). Not needed when using an external HSS.
- `ralf_hostname` - a hostname that resolves by DNS round-robin to the signaling interface of all Dime nodes in the cluster. Should include the port (usually 9888). This is also used (without the port) as the Origin-Realm of the Diameter messages the ralf process on Dime sends. Optional if ralf is not being used.
- `cdf_identity` - a Diameter identity that represents the address of an online Charging Function. Subscribers provisioned through Ellis will have this set as their Primary Charging Collection Function on P-Charging-Function-Addresses headers on responses to their successful REGISTERs, and Bono will add similarly in originating requests.

- xdms_hostname - a hostname that resolves by DNS round-robin to all Homer nodes in the cluster. Should include the port (usually 7888).
- hss_realm - this sets the Destination-Realm of your external HSS. When this field is set, the homestead process on Dime will then attempt to set up multiple Diameter connections using an SRV lookup on this realm.
- hss_hostname - this sets the Destination-Host of your external HSS, if you have one. The homestead process on Dime will also try and establish a Diameter connection to this host (on port 3868) if no SRV-discovered peers exist.
- signup_key - this sets the password which Ellis will require before allowing self-sign-up.
- turn_workaround - if your STUN/TURN clients are not able to authenticate properly (for example, because they can't send the @ sign), this specifies an additional password which will authenticate clients even without a correct username.
- smtp_smarthost - Ellis allows password recovery by email. This sets the SMTP server used to send those emails.
- smtp_username - Ellis allows password recovery by email. This sets the username used to log in to the SMTP server.
- smtp_password - Ellis allows password recovery by email. This sets the password used to log in to the SMTP server.
- email_recovery_sender - Ellis allows password recovery by email. This sets the email address those emails are sent from.
- ellis_api_key - sets a key which can be used to authenticate automated requests to Ellis, by setting it as the value of the X-NGV-API header. This is used to expire demo users regularly.
- ellis_hostname - a hostname that resolves to Ellis, if you don't want to use ellis.home_domain. This should match Ellis's SSL certificate, if you are using one.
- memento_hostname - a hostname that resolves by DNS round-robin to all Mementos in the cluster (the default is memento.<home_domain>). This should match Memento's SSL certificate, if you are using one.
- sprout_registration_store - this is the location of Sprout's registration store. It has the format <site_name>=<domain>[:<port>][,<site_name>=<domain>[:<port>]]. In a non-GR deployment, only one domain is provided (and the site name is optional). For a GR deployment, each domain is identified by the site name, and one of the domains must relate to the local site.
- ralf_session_store - this is the location of ralf's session store. It has the format <site_name>=<domain>[:<port>][,<site_name>=<domain>[:<port>]]. In a non-GR deployment, only one domain is provided (and the site name is optional). For a GR deployment, each domain is identified by the site name, and one of the domains must relate to the local site.
- homestead_impu_store - this is the location of homestead's IMPU store. It has the format <site_name>=<domain>[:<port>][,<site_name>=<domain>[:<port>]]. In a non-GR deployment, only one domain is provided (and the site name is optional). For a GR deployment, each domain is identified by the site name, and one of the domains must relate to the local site.
- memento_auth_store - this is the location of Memento's authorization vector store. It just has the format <domain>[:port]. If not present, defaults to the loopback IP.
- sprout_chronos_callback_uri - the callback hostname used on Sprout's Chronos timers. If not present, defaults to the host specified in sprout-hostname. In a GR deployment, should be set to a deployment-wide Sprout hostname (that will be resolved by using static DNS records in /etc/clearwater/dns.json).

- ralf_chronos_callback_uri - the callback hostname used on ralf's Chronos timers. If not present, defaults to the host specified in ralf-hostname. In a GR deployment, should be set to a deployment-wide Dime hostname (that will be resolved by using static DNS records in /etc/clearwater/dns.json).
- cassandra_hostname - a hostname that resolves by DNS round-robin to the signaling interface of all Vellum nodes in the local site.
- chronos_hostname - a hostname that resolves by DNS round-robin to the signaling interface of all Vellum nodes in the local site.

Sproutlet options

This section describes optional configuration options for the Clearwater Sproutlets. Sproutlets are built on top of Sprout, and encapsulate the business logic of the I-CSCF/S-CSCF/BGCF, or Project Clearwater's built in Application servers

There are currently eight different Sproutlets:

- S-CSCF - Provides S-CSCF functionality
- I-CSCF - Provides I-CSCF functionality
- BGCF - Provides BGCF functionality
- Gemini - An application server responsible for twinning VoIP clients with a mobile phone hosted on a native circuit-switched network. You can find out more here
- Memento - An application server responsible for providing network-based call lists. You can find out more here
- CDiv - Provides call diversion functionality
- MMtel - Acts as a basic MMTel AS
- Mangelwurzel - Acts as a basic B2BUA

Each Sproutlet has three configuration options. The options have the same format for each Sproutlet, as listed here, with <sproutlet> replaced by the appropriate Sproutlet name:

- <sproutlet> - The port that the Sproutlet listens on. The default value depends on the Sproutlet. Some Sproutlets default to 0 (meaning that they are disabled by default). For other Sproutlets, the defaults are:

```
I-CSCF - 5052
BGCF - 5053
S-CSCF - 5054
MMTel - 5055
```

- <sproutlet>_prefix - The identifier prefix for this Sproutlet, used to build the uri, as described below. The default value is simply the Sproutlet name: <sproutlet>
- <sproutlet>_uri - The full identifier for this Sproutlet, used for routing and receiving requests between nodes. The default value is created using the prefix and the hostname of the parent Sprout node, i.e. sip:<sproutlet_prefix>.<sprout_hostname>;transport=tcp. We recommend that you don't set this yourself anymore, and use the defaults provided.

As a concrete example, below are the S-CSCF options and the default values.

- scscf=5054
- scscf_prefix=scscf
- scscf_uri=sip:scscf.<sprout_hostname>;transport=tcp

Advanced options

This section describes optional configuration options, particularly for ensuring conformance with other IMS devices such as HSSes, ENUM servers, application servers with strict requirements on Record-Route headers, and non-Clearwater I-CSCFs. These options should be set in the `/etc/clearwater/shared_config` file (in the format `name=value`, e.g. `icscf=5052`).

- `homestead_provisioning_port` - the HTTP port the homestead provisioning interface on Dime listens on. Defaults to 8889. Not needed when using an external HSS.
- `sas_server` - the IP address or hostname of your Metaswitch Service Assurance Server for call logging and troubleshooting. Optional.
- `reg_max_expires` - determines the maximum expires= parameter Sprout will set on Contact headers at registrations, and therefore the amount of time before a UE has to re-register - must be less than 2^31 ms (approximately 25 days). Default is 300 (seconds).
- `sub_max_expires` - determines the maximum Expires header Sprout will set in subscription responses, and therefore the amount of time before a UE has to re-subscribe - must be less than 2^31 ms (approximately 25 days).
- `upstream_hostname` - the I-CSCF which Bono should pass requests to. Defaults to `icscf.<sprout_hostname>`.
- `upstream_port` - the port on the I-CSCF which Bono should pass requests to. Defaults to 5052. If set to 0, Bono will use SRV resolution of the `upstream_hostname` hostname to determine a target for traffic.
- `sprout_rr_level` - this determines how the Sprout S-CSCF adds Record-Route headers. Possible values are:
  - `pcscf` - a Record-Route header is only added just after requests come from or go to a P-CSCF - that is, at the start of originating handling and the end of terminating handling
  - `pcscf,icscf` - a Record-Route header is added just after requests come from or go to a P-CSCF or I-CSCF - that is, at the start and end of originating handling and the start and end of terminating handling
  - `pcscf,icscf,as` - a Record-Route header is added after requests come from or go to a P-CSCF, I-CSCF or application server - that is, at the start and end of originating handling, the start and end of terminating handling, and between each application server invoked
- `force_hss_peer` - when set to an IP address or hostname, the homestead process on Dime will create a connection to the HSS using this value, but will still use the `hss_realm` and `hss_hostname` settings for the Destination-Host and Destination-Realm Diameter AVPs. This is useful when your HSS's Diameter configuration does not match the DNS records.
- `hss_mar_scheme_unknown` - if Clearwater cannot tell what authentication type a subscriber is trying to use, this field determines what authentication scheme it requests in the Multimedia-Auth-Request. Default value is 'Unknown'.
- `hss_mar_scheme_digest` - if Clearwater determines a subscriber is trying to use password-based digest authentication, this field determines what authentication scheme it requests in the Multimedia-Auth-Request. Default value is 'SIP Digest'.
- `hss_mar_scheme_akav1` - if Clearwater determines a subscriber is trying to use AKAv1 authentication, this field determines what authentication scheme it requests in the Multimedia-Auth-Request. Default value is 'Digest-AKAv1-MD5'.
- `hss_mar_scheme_akav2` - if Clearwater determines a subscriber is trying to use AKAv2 authentication, this field determines what authentication scheme it requests in the Multimedia-Auth-Request. Default value is 'Digest-AKAv2-SHA-256'.

- `force_third_party_reg_body` - if the HSS does not allow the IncludeRegisterRequest/IncludeRegisterResponse fields (which were added in 3GPP Rel 9) to be configured, setting `force_third_party_reg_body=Y` makes Clearwater behave as though they had been sent, allowing interop with application servers that need them.
- `enforce_user_phone` - by default, Clearwater will do an ENUM lookup on any SIP URI that looks like a phone number, due to client support for user-phone not being widespread. When this option is set to 'Y', Clearwater will only do ENUM lookups for URIs which have the user=phone parameter.
- `enforce_global_only_lookups` - by default, Clearwater will do ENUM lookups for SIP and Tel URIs containing global and local numbers (as defined in RFC 3966). When this option is set to 'Y', Clearwater will only do ENUM lookups for SIP and Tel URIs that contain global numbers.
- `hs_listen_port` - the Diameter port on which the homestead process on Dime listens. Defaults to 3868.
- `ralf_listen_port` - the Diameter port on which the ralf process on Dime listens. Defaults to 3869 to avoid clashes with the homestead process.
- `alias_list` - this defines additional hostnames and IP addresses which Sprout or Bono will treat as local for the purposes of SIP routing (e.g. when removing Route headers).
- `bono_alias_list` - this defines additional hostnames and IP addresses specifically for Bono which will be treated as local for the purposes of SIP routing.
- `default_session_expires` - determines the Session-Expires value which Sprout will add to INVITEs, to force UEs to send keepalive messages during calls so they can be tracked for billing purposes. This cannot be set to a value less than 90 seconds, as specified in RFC 4028, section 4.
- `max_session_expires` - determines the maximum Session-Expires/Min-SE value which Sprout will accept in requests. This cannot be set to a value less than 90 seconds, as specified in RFC 4028, sections 4 and 5.
- `enum_server` - a comma-separated list of DNS servers which can handle ENUM queries.
- `enum_suffix` - determines the DNS suffix used for ENUM requests (after the digits of the number). Defaults to "e164.arpa"
- `enum_file` - if set (to a file path), and if `enum_server` is not set, Sprout will use this local JSON file for ENUM lookups rather than a DNS server. An example file is on our ENUM page.
- `external_icscf_uri` - the SIP address of the external I-CSCF integrated with your Sprout node (if you have one).
- `additional_home_domains` - this option defines a set of home domains which Sprout and Bono will regard as locally hosted (i.e. allowing users to register, not routing calls via an external trunk). It is a comma-separated list.
- `billing_realm` - when this field is set, the ralf process on Dime will attempt to set up multiple Diameter connections using an SRV lookup on this realm. Messages sent on these connections will have:
  - Destination-Realm set to the `billing_realm` value
  - Destination-Host set to the value of the `ccf` parameter in the P-Charging-Function-Addresses SIP header received from the P-CSCF, or from the Primary-Charging-Collection-Function-Name/Secondary-Charging-Collection-Function-Name AVPs received over the Cx interface from the HSS.
- `diameter_timeout_ms` - determines the number of milliseconds homestead will wait for a response from the HSS before failing a request. Defaults to 200.
- `max_peers` - determines the maximum number of Diameter peers to which the ralf or homestead processes on Dime can have open connections at the same time.

- num_http_threads (ralf/memento) - determines the number of threads that will be used to process HTTP requests. For memento this defaults to the number of CPU cores on the system. For ralf it defaults to 50 times the number of CPU cores (memento and ralf use different threading models, hence the different defaults). Note that for homestead, this can only be set in /etc/clearwater/user_settings.
- num_http_worker_threads - determines the number of threads that will be used to process HTTP requests once they have been parsed. Only used by Memento.
- ralf_diameteridentity - determines the Origin-Host that will be set on the Diameter messages ralf sends. Defaults to public_hostname (with some formatting changes if public_hostname is an IPv6 address).
- hs_diameteridentity - determines the Origin-Host that will be set on the Diameter messages homestead sends. Defaults to public_hostname (with some formatting changes if public_hostname is an IPv6 address).
- max_call_list_length - determines the maximum number of complete calls a subscriber can have in the call list store. This defaults to no limit. This is only relevant if the node includes a Memento AS.
- call_list_store_ttl - determines how long each call list fragment should be kept in the call list store. This defaults to 604800 seconds (1 week). This is only relevant if the node includes a Memento AS.
- memento_disk_limit - determines the maximum size that the call lists database may occupy. This defaults to 20% of disk space. This is only relevant if the node includes a Memento AS. Can be specified in Bytes, Kilobytes, Megabytes, Gigabytes, or a percentage of the available disk. For example:

```
memento_disk_limit=10240 # Bytes
memento_disk_limit=100k  # Kilobytes
memento_disk_limit=100M  # Megabytes
memento_disk_limit=100G  # Gigabytes
memento_disk_limit=45%   # Percentage of available disk
```

- memento_threads - determines the number of threads dedicated to adding call list fragments to the call list store. This defaults to 25 threads. This is only relevant if the node includes a Memento AS.
- memento_notify_url - If set to an HTTP URL, memento will make a POST request to this URL whenever a subscriber's call list changes. The body of the POST request will be a JSON document with the subscriber's IMPU in a field named impu. This is only relevant if the node includes a Memento AS. If empty, no notifications will be sent. Defaults to empty.
- signaling_dns_server - a comma-separated list of DNS servers for non-ENUM queries. Defaults to 127.0.0.1 (i.e. uses dnsmasq)
- target_latency_us - Target latency (in microsecs) for requests above which throttling applies. This defaults to 100000 microsecs
- max_tokens - Maximum number of tokens allowed in the token bucket (used by the throttling code). This defaults to 1000 tokens
- init_token_rate - Initial token refill rate of tokens in the token bucket (used by the throttling code). This defaults to 250 tokens per second per core
- min_token_rate - Minimum token refill rate of tokens in the token bucket (used by the throttling code). This defaults to 10.0
- override_npdi - Whether the I-CSCF, S-CSCF and BGCF should check for number portability data on requests that already have the 'npdi' indicator. This defaults to false
- exception_max_ttl - determines the maximum time before a process exits if it crashes. This defaults to 600 seconds
- check_destination_host - determines whether the node checks the Destination-Host on a Diameter request when deciding whether it should process the request. This defaults to true.

- astaire_cpu_limit_percentage - the maximum percentage of total CPU that Astaire is allowed to consume when resyncing memcached data (as part of a scale-up, scale-down, or following a memcached failure). Note that this only limits the CPU usage of the Astaire process, and does not affect memcached's CPU usage. Must be an integer. Defaults to 5.
- sip_blacklist_duration - the time in seconds for which SIP peers are blacklisted when they are unresponsive (defaults to 30 seconds).
- http_blacklist_duration - the time in seconds for which HTTP peers are blacklisted when they are unresponsive (defaults to 30 seconds).
- diameter_blacklist_duration - the time in seconds for which Diameter peers are blacklisted when they are unresponsive (defaults to 30 seconds).
- snmp_ip - the IP address to send alarms to (defaults to being unset). If this is set then Sprout, Dime and Vellum will send alarms - more details on the alarms are here. This can be a single IP address, or a comma-separated list of IP addresses.
- snmp_notification_types - this determines what format SNMP alarms are sent in, and is a comma-separated list of SNMP alarm formats. Valid alarm formats are rfc3877 and enterprise - if both are set, every alarm generates two SNMP INFORMs, one in each format . See the SNMP alarms documentation for information about the difference between the formats.
- impu_cache_ttl - the number of seconds for which homestead will cache the SIP Digest from a Multimedia-Auth-Request. Defaults to 0, as Sprout does enough caching to ensure that it can handle an authenticated REGISTER after a challenge, and subsequent challenges should be rare.
- sip_tcp_connect_timeout - the time in milliseconds to wait for a SIP TCP connection to be established (defaults to 2000 milliseconds).
- sip_tcp_send_timeout - the time in milliseconds to wait for sent data to be acknowledgered at the TCP level on a SIP TCP connection (defaults to 2000 milliseconds).
- session_continued_timeout_ms - if an Application Server with default handling of 'continue session' is unresponsive, this is the time that Sprout will wait (in milliseconds) before bypassing the AS and moving onto the next AS in the chain (defaults to 2000 milliseconds).
- session_terminated_timeout_ms - if an Application Server with default handling of 'terminate session' is unresponsive, this is the time that Sprout will wait (in milliseconds) before terminating the session (defaults to 4000 milliseconds).
- sas_use_signaling_interface - When this field is set to 'Y', SAS traffic is routed via the signaling network, rather than the management network.
- pbxes - a comma separated list of IP address that Bono considers to be PBXes that are incapable of registering. Non-REGISTER requests from these addresses are passed upstream to Sprout with a Proxy-Authorization header. It is strongly recommended that Sprout's non_register_authentication option is set to if_proxy_authorization_present so that the request will be challenged. Bono also permits requests to these addresses from the core to pass through it.
- pbx_service_route - the SIP URI to which Bono routes originating calls from non-registering PBXes (which are identified by the pbxes option). This is used to route requests directly to the S-CSCF rather than going via an I-CSCF (which could change the route header and prevent the S-CSCF from processing the request properly). This URI is used verbatim and should almost always include the lr, orig, and auto-reg parameters. If this option is not specified, the requests are routed to the address specified by the upstream_hostname and upstream_port options.
  - e.g. sip:sprout.example.com:5054;transport=tcp;lr;orig;auto-reg
- non_register_authentication - controls when Sprout will challenge a non-REGISTER request using SIP Proxy-Authentication. This option is a comma separated list that may contain the val-

ues listed below (e.g. `non_register_authentication=if_proxy_authorization_present, initial_req_from_req_digest_endpoint`):

- `if_proxy_authorization_present`: Sprout will authenticate requests that have a Proxy-Authorization header.
- `initial_req_from_reg_digest_endpoint`: Sprout will authenticate requests from registered endpoints that use the SIP digest authentication scheme.
- `ralf_threads` - used on Sprout nodes, this determines how many worker threads should be started to do ralf request processing (defaults to 25).
- `impi_store_mode` - used to control how Sprout stores authentication challenges. The default is `impi` which means that challenges are written to a single memcached database table indexed by IMPI. There is another option, `av-impi`, where challenges are also stored in an old table indexed by (IMPI, nonce). This setting can be used to upgrade Clearwater to use the new database table without losing registration state.
- `nonce_count_supported` - when set to 'Y' Clearwater permits authentication responses with a nonce-count greater than 1. By default this option is not enabled. Enabling this option can expose certain security holes if your deployment does not use an HSS (and uses Homestead-Prov instead) and an I-CSCF. Specifically if the option is set and a malicious UE manages to register:
  - Without an HSS there is no way to force it to become deregistered.
  - Without an I-CSCF there is no way to prevent it from registering as different user accounts.
- `disable_tcp_switch` - when set to 'Y', Clearwater disables UDP-to-TCP uplift on SIP messages. This is useful when creating a deployment where all SIP is sent over UDP. This option only affects Sprout nodes.
- `sprout_impi_store` - this is the location of Sprout's IMPI store. It has the same format as `sprout_registration_store`. If not provided, Sprout uses the same value configured in `sprout_registration_store`.
- `request_shared_ifcs` - when set to 'Y' Clearwater requests Shared iFC sets from the HSS. Shared iFC sets can be configured on Clearwater in the `/etc/clearwater/shared_ifcs.xml` file. This option is not enabled by default.
- `apply_fallback_ifcs` - when set to 'Y' Clearwater will apply any fallback iFCs specified by the operator in the `/etc/clearwater/fallback_ifcs.xml` file to initial requests who have no applicable iFCs associated with them. This option is not enabled by default.
- `reject_if_no_matching_ifcs` - when set to 'Y' Clearwater will reject any initial requests that don't have any matching iFCs that can be applied to them. This option is not enabled by default.
- `dummy_app_server` - this field allows the name of a dummy application server to be specified. If an iFC contains this dummy application server, then no application server will be invoked when this iFC is triggered.
- `http_acr_logging` when set to 'Y', Clearwater will log the bodies of HTTP requests made to Ralf. This provides additional diagnostics, but increases the volume of data sent to SAS.
- `dns_timeout` - The time in milliseconds that Clearwater will wait for a response from the DNS server (defaults to 200 milliseconds).
- `homestead_cache_threads` - The number of threads used by Homestead for accessing it's subscriber data cache. Defaults to 50x the number of CPU cores.

Experimental options

This section describes optional configuration options which may be useful, but are not heavily-used or well-tested by the main Clearwater development team. These options should be set in the `/etc/clearwater/shared_config` file (in the format `name=value`, e.g. `ralf_secure_listen_port=12345`).

- ralf_secure_listen_port - this determines the port the ralf process on Dime listens on for TLS-secured Diameter connections.
- hs_secure_listen_port - this determines the port the homestead process on Dime listens on for TLS-secured Diameter connections.
- ellis_cookie_key - an arbitrary string that enables Ellis nodes to determine whether they should be in the same cluster. This function is not presently used.
- stateless_proxies - a comma separated list of domain names that are treated as SIP stateless proxies. Stateless proxies are not blacklisted if a SIP transaction sent to them times out. This field should reflect how the servers are identified in SIP. For example if a cluster of nodes is identified by the name 'cluster.example.com', the option should be set to 'cluster.example.com' instead of the hostnames or IP addresses of individual servers.
- hss_reregistration_time - determines how many seconds should pass before homestead sends a Server-Assignment-Request with type RE_REGISTRATION to the HSS. (On first registration, it will always send a SAR with type REGISTRATION). This determines a minimum value - after this many seconds have passed, homestead will send the Server-Assignment-Request when the next REGISTER is received. Note that homestead invalidates its cache of the registration and iFCs after twice this many seconds have passed, so it is not safe to set this to less than half of reg_max_expires. The default value of this option is whichever is the greater of the following.
  - 1800.
  - Half of the value of reg_max_expires.

User settings

This section describes settings that may vary between systems in the same deployment, such as log level (which may be increased on certain machines to track down specific issues) and performance settings (which may vary if some servers in your deployment are more powerful than others). These settings are set in /etc/clearwater/user_settings (in the format name=value, e.g. log_level=5).

- log_level - determines how verbose Clearwater's logging is, from 1 (error logs only) to 5 (debug-level logs). Defaults to 2.
- log_directory - determines which folder the logs are created in. This folder must exist, and be owned by the service. Defaults to /var/log/ (this folder is created and has the correct permissions set for it by the install scripts of the service).
- max_log_directory_size - determines the maximum size of each Clearwater process's log_directory in bytes. Defaults to 1GB. If you are co-locating multiple Clearwater processes, you'll need to reduce this value proportionally.
- num_worker_threads - for Sprout and Bono nodes, determines how many worker threads should be started to do SIP/IMS processing. Defaults to 50 times the number of CPU cores on the system.
- upstream_connections - determines the maximum number of TCP connections which Bono will open to the I-CSCF(s). Defaults to 50.
- trusted_peers - For Bono IBCF nodes, determines the peers which Bono will accept connections to and from.
- ibcf_domain - For Bono IBCF nodes, allows for a domain alias to be specified for the IBCF to allow for including IBCFs in routes as domains instead of IPs.
- upstream_recycle_connections - the average number of seconds before Bono will destroy and re-create a connection to Sprout. A higher value means slightly less work, but means that DNS changes will not take effect as quickly (as new Sprout nodes added to DNS will only start to receive messages when Bono creates a new connection and does a fresh DNS lookup).

- `authentication` - by default, Clearwater performs authentication challenges (SIP Digest or IMS AKA depending on HSS configuration). When this is set to 'Y', it simply accepts all REGISTERs - obviously this is very insecure and should not be used in production.
- `num_http_threads` (homestead) - determines the number of HTTP worker threads that will be used to process requests. Defaults to 4 times the number of CPU cores on the system.

DNS Config

This section describes the static DNS config which can be used to override DNS results. This is set in `/etc/clearwater/dns.json`. Currently, the only supported record type is CNAME and the only component which uses this is Chronos and the I-CSCF. The file has the format:

```
{
  "hostnames": [
    {
      "name": "<hostname 1>",
      "records": [{"rrtype": "CNAME", "target": "<target for hostname 1>"}]
    },
    {
      "name": "<hostname 2>",
      "records": [{"rrtype": "CNAME", "target": "<target for hostname 2>"}]
    }
  ]
}
```

Other configuration options

There is further documentation for Chronos configuration here and Homer/Homestead-prov configuration here.

Configuring an Application Server

This document explains how to configure an application server for use in a Clearwater deployment.

Network configuration

Each application server must be able to communicate with Sprout and Bono in both directions.

- The application server must be able to contact all sprout and bono nodes on the trusted SIP ports, 5054 and 5058 (respectively, for both TCP and UDP).
- All sprout and bono nodes must be able to contact the application server on the port and protocol configured in your subscribers' iFCs (typically TCP and UDP port 5060).
- The application server must also be able to exchange SIP messages (in both directions) with any other application server that may appear in the signaling path of any call going through it.
- Since the application server is in the trusted zone, it should *not* be accessible to untrusted SIP entities.

If Clearwater and your application server are both in the Amazon EC2 cloud, you can achieve all of these by placing the application server in the `<deployment>-sprout` security group.

Application server configuration

No special application server configuration is required.

- Your application server should be prepared to accept SIP messages from any of your deployment's bono or sprout nodes, and from any SIP entity that may appear in the signaling path of any call going through it.
- If your application server needs to spontaneously originate calls, it should do this via Sprout's trusted interface: `sprout.<deployment-domain>:5054` over either TCP or UDP.

The headers set by Clearwater are described in the Application Server Guide.

iFC configuration

To configure a subscriber to invoke your application server, you must configure the appropriate iFCs for that subscriber. You can do this via the Ellis UI, the Homestead API, or if you are using an HSS, directly in the HSS.

Web UI configuration via Ellis

Ellis allows you to specify a mapping between application server names and XML nodes. This is done by editing the `/usr/share/clearwater/ellis/web-content/js/app-servers.json` file, which is in JSON format. An example file would be:

```
{
"MMTEL" : "<InitialFilterCriteria><Priority>0</Priority><TriggerPoint>
↪<ConditionTypeCNF></ConditionTypeCNF><SPT><ConditionNegated>0</ConditionNegated>
↪<Group>0</Group><Method>INVITE</Method><Extension></Extension></SPT></TriggerPoint>
↪<ApplicationServer><ServerName>sip:mmtel.example.com</ServerName><DefaultHandling>0
↪</DefaultHandling></ApplicationServer></InitialFilterCriteria>",
"Voicemail" : "<InitialFilterCriteria><Priority>1</Priority><TriggerPoint>
↪<ConditionTypeCNF></ConditionTypeCNF><SPT><ConditionNegated>0</ConditionNegated>
↪<Group>0</Group><Method>INVITE</Method><Extension></Extension></SPT></TriggerPoint>
↪<ApplicationServer><ServerName>sip:vm.example.com</ServerName><DefaultHandling>0</
↪DefaultHandling></ApplicationServer></InitialFilterCriteria>"
}
```

Once this is saved, the list of application server names will appear in the Ellis UI (on the 'Application Servers' tab of the Configure dialog), and selecting or deselecting them will add or remove the relevant XML from Homestead. This change takes effect immediately. If an node in the iFC XML is not included in app-servers.json, Ellis will leave it untouched.

Direct configuration via cURL

You can also configure iFCs directly using Homestead.

Here is an example of how to use `curl` to configure iFCs directly. This configures a very basic iFC, which fires on INVITEs only, with no conditions on session case. See an iFC reference for more details, e.g., 3GPP TS 29.228 appendices B and F.

To simplify the following commands, define the following variables - set the user, application server(s), and Homestead name as appropriate for your deployment.

```
user=sip:6505550269@example.com
as_hostnames="as1.example.com mmtel.example.com" # the list of zero or more␣
↪application servers to invoke - don't forget mmtel
hs_hostname=hs.example.com:8888
```

Be careful - the user must be *exactly* right, including the `sip:` prefix, and also `+1` if and only if it is a PSTN line. We have seen problems with PSTN lines; if the above syntax does not work, try URL-encoding the user, e.g., for +16505550269@example.com write `user=sip%3A%2B16505550269%40example.com`.

To retrieve the current configuration, invoke `curl` as follows. You must be able to access $hs_hostname; check your firewall configuration.

```
curl http://$hs_hostname:8889/public/$user/service_profile
```

This will return a 303 if the user exists, with the service profile URL in the Location header, e.g.

```
Location: /irs/<irs-uuid>/service_profiles/<service-profile-uuid>
```

Then invoke `curl` as follows, using the values from the Location header retrieved above:

```
curl http://$hs_hostname:8889/irs/<irs-uuid>/service_profiles/<service-profile-uuid>/
↪filter_criteria
```

To update the configuration, invoke `curl` as follows. The first stage builds the new XML document and the second pushes it to the server.

```
{ cat <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<ServiceProfile>
EOF
for as_hostname in $as_hostnames ; do
  cat <<EOF
  <InitialFilterCriteria>
    <Priority>1</Priority>
    <TriggerPoint>
      <ConditionTypeCNF>0</ConditionTypeCNF>
      <SPT>
        <ConditionNegated>0</ConditionNegated>
        <Group>0</Group>
        <Method>INVITE</Method>
        <Extension></Extension>
      </SPT>
    </TriggerPoint>
    <ApplicationServer>
      <ServerName>sip:$as_hostname</ServerName>
      <DefaultHandling>0</DefaultHandling>
    </ApplicationServer>
  </InitialFilterCriteria>
EOF
done
cat <<EOF
</ServiceProfile>
EOF
} | curl -X PUT http://$hs_hostname:8889/irs/<irs-uuid>/service_profiles/<service-
↪profile-uuid>/filter_criteria --data-binary @-
```

The subscriber will now have the desired configuration. You can confirm this by running the retrieval command again.

SIP-over-UDP configuration

While our default configuration for Clearwater deployments is to send SIP over TCP, it is possible to create a SIP-over-UDP deployment, with certain restrictions.

Restrictions

If you want to create a SIP-over-UDP deployment, it will be necessary for all of Sprout's SIP peers to send SIP over UDP to it. It is not possible to set up some peers to use TCP and some to use UDP. This is because Sprout won't do UDP/TCP interworking.

Configuration

To enable SIP-over-UDP, you will need to set the following configuration options.

In `/etc/clearwater/shared_config` set or update the fields:

```
scscf_uri="sip:scscf.<sprout_hostname>;transport=udp"
disable_tcp_switch=Y
```

In `/etc/clearwater/local_config` set or update the field on each of your Sprout nodes:

```
scscf_node_uri="sip:<local_ip>:5054;transport=udp"
```

You may also need to:

- Set up your P-CSCF and any application servers to send SIP over UDP.
- Add new SRV entries to your DNS server for your Application Server.
- Add `transport=udp` to your Application Server's SIP URI on your HSS.

You should now be able to make calls where SIP is sent over UDP.

External HSS Integration

All Clearwater deployments include a cluster of Dime nodes that run the Homestead service. Homestead presents an HTTP RESTful interface to HSS data. This HSS data can be stored in either

- a Cassandra database located on the Vellum cluster
- an external HSS, accessible over a standard IMS Cx/Diameter interface.

This page describes

- how it works
- how to enable it
- restrictions.

How It Works

When Clearwater is deployed without an external HSS, all HSS data is mastered in Vellum's Cassandra database.

When Clearwater is deployed with an external HSS, HSS data is queried from the external HSS via its Cx/Diameter interface and is then cached in Memcached on Vellum.

Clearwater uses the following Cx message types.

- Multimedia-Auth - to retrieve authentication details
- Server-Assignment - to retrieve Initial Filter Criteria documents and register for change notifications

- Push-Profile - to be notified of changes to Initial Filter Criteria documents
- User-Authorization - to retrieve S-CSCF details on initial registrations
- Location-Information - to retrieve S-CSCF details on calls
- Registration-Termination - to be notified of de-registrations

How to Enable It

This section discusses how to enable support for an external HSS.

Before you start

Before enabling support for an external HSS, you must

- install Clearwater
- install an external HSS - details for this will vary depending on which HSS you choose, but there are instructions for OpenIMSCore HSS.

Do not configure any Clearwater subscribers via Ellis!

- Any subscribers you create before enabling the external HSS will override subscribers retrieved from the external HSS, and you will not be able to use Ellis to manage them.
- After enabling the external HSS, you will not be able to create subscribers through Ellis at all - they must be created through the HSS's own management interface.

Enabling external HSS support on an existing deployment

To enable external HSS support, you will need to modify the contents of `/etc/clearwater/shared_config` so that the block that reads

```
# HSS configuration
hss_hostname=0.0.0.0
hss_port=3868
```

instead reads

```
# HSS configuration
hss_hostname=<address of your HSS>
hss_realm=<realm your HSS is located in>
hss_port=<port of your HSS's Cx interface>
```

Both hss_hostname and hss_realm are optional. If a realm is configured, homestead will try NAPTR/SRV resolution on the realm to find and connect to (2 by default) diameter peers in the realm. If a hostname is also configured, this will be used in the Destination-Host field on the diameter messages, so that the messages will be routed to that host. If just a hostname is configured, homestead will just attempt to create and maintain a single connection to that host.

This process explains how to modify these settings and ensure that all nodes pick up the changes.

Configuring your external HSS

Homestead will now query the external HSS for subscriber details when they are required by the rest of the Clearwater deployment, such as when servicing SIP registrations.

In order to register and make calls, you need to create subscriber records on your external HSS, and the detailed process for this will depend on which HSS you have chosen. Generally, however, you will need to

1. create a public user identity with the desired SIP URI
2. create an associated private user identity with its ID formed by removing the `sip:` scheme prefix from the public user ID
3. configure the public user identity's Initial Filter Criteria to include an application server named `sip:mmtel.your.home.domain`, where `your.home.domain` is replaced with your home domain - this enables MMTEL services for this subscriber.

Allowing one subscriber to have two private identities

If you try to use an Android SIP client that doesn't contain an **Authentication username** field, the client will default to a username like **''1234''** (rather than **''1234@example.com''** - the IMS standard form). To register a subscriber you will have to configure your external HSS so that the subscriber you are trying to register has two private identities (**''1234''** and **''1234@example.com''**).

The detailed process for this will depend on which HSS you have chosen. Generally, however, you will need to

1. create a subscriber as usual- with public user identity "sip:<*username*>@<*server*>"
2. create two new private identities, one having identity "<*username*>" and the other with identity "<*username*>@<*server*>"
3. associate the public user identity that was created in step 1: "sip:<*username*>@<*server*>" to both of the private identities created in step 2.

This should allow the SIP client to register with that subscriber.

Restrictions

- Since Homestead uses the Cx/Diameter interface to the HSS, and this interface is read-only, the Homestead API is read-only when external HSS integration is enabled.
- Since Homestead's API is read-only, this means that Ellis can't be used alongside a deployment using an external HSS. Provisioning and subscriber management must be performed via the HSS's own management interface.
- Clearwater currently only supports SIP digest or AKA authentication; details for other authentication methods are silently ignored from Multimedia-Auth responses. Other authentication methods may be added in future.
- Homestead currently assumes that private user IDs are formed by removing the `sip:` prefix from the public user ID. This restriction may be relaxed in future.
- While Homestead caches positive results from the external HSS, it does not currently cache negative results (e.g. for non-existent users). Repeated requests for a non-existent user will increase the load on the external HSS. This restriction may be relaxed in future.

OpenIMS HSS Integration

As discussed on the External HSS Integration page, Clearwater can integrate with an external HSS.

This page describes how to install and configure the OpenIMSCore HSS as this external HSS. It assumes that you have already read the External HSS Integration page.

Installation with Chef

If you have a deployment environment created by following the automated install instructions, then you can create a HSS by running `knife box create -E <env> openimscorehss`. You should then follow the configuration instructions below.

Installing OpenIMSCore HSS manually

To install OpenIMSCore HSS,

1. Follow the "Adding this PPA to your system" steps at https://launchpad.net/~rkd-u/+archive/ubuntu/fhoss
2. Run `sudo apt-get update; sudo apt-get install openimscore-fhoss`
3. Answer the installation questions appropriately, including providing the IMS home domain, your local IP, and the MySQL root password (which you will have to provide both when installing MySQL and when intalling the HSS).

Configuration

Logging in

OpenIMSCore HSS provides the administration UI over port 8080. The admin username is hssAdmin.

- If the HSS was installed using Chef, the hssAdmin password will be the signup_key setting from knife.rb.
- If the HSS was installed manually, the hssAdmin password will be "hss". This can be changed by editing `/usr/share/java/fhoss-0.2/conf/tomcat-users.xml` and running `sudo service openimscore-fhoss restart`.

Adding the MMTEL Application Server

To enable the MMTEL application server built into Clearwater for all subscribers,

1. access the OpenIMSCore HSS web UI, running on port 8080
2. log in as hssAdmin
3. navigate to `Services`->``Application Servers``->``Search``
4. perform a search with no search string to find all application servers
5. select default_as
6. change `server_name` to be `sip:mmtel.your.home.domain`, where `your.home.domain` is replaced with your home domain
7. save your change.

Configuring Subscribers

To configure a subscriber,

1. create an IMS subscription
2. create an associated public user identity
3. create an associated private user identity, specifying

   (a) the private ID to be the public ID without the `sip:` scheme prefix

   (b) SIP Digest authentication.

4. add the home domain to the subscriber's visited networks

CDF Integration

Project Clearwater deployments include a cluster of Dime nodes running Ralf. The nodes provide an HTTP interface to Sprout and Bono on which they can report billable events. Ralf then acts as a CTF (Charging Triggering Function) and may pass these events on to a configured CDF (Charging Data Function) over the Rf interface.

By default, having spun up a Clearwater deployment, either manually or through the automated Chef install process, your deployment is not configured with a CDF and thus will not generate billing records. This document describes how the CDF is chosen and used and how to integrate your Project Clearwater deployment with a CDF and thus enable Rf billing.

How it works

When Sprout or Bono have handled an item of work for a given subscriber, they generate a record of that work and transmit it to the Dime cluster for billing to the CDF. This record will be used by Ralf to generate an Rf billing ACR (Accounting-Request) message.

To determine which CDF to send the ACR to, the node acting as the P-CSCF/IBCF is responsible for adding a `P-Charging-Function-Addresses` header to all SIP messages it proxies into the core. This header contains a prioritised list of CDFs to send ACRs to.

Bono supports adding this header when acting as either a P-CSCF or IBCF and configuring the contents of this header is the only requirement for enabling Rf billing in your deployment.

How to enable it

This section discusses how to enable Rf billing to a given CDF.

Before you begin

Before connecting your deployment to a CDF, you must

- install Clearwater
- install an external CDF - details for this will vary depending on which CDF you are using.
- ensure your CDF's firewall allows incoming connections from the nodes in the Dime cluster on the DIAMETER port (default 3868).

Setting up DNS

Ralf implements the behavior specified in RFC3588 to locate and connect to the billing realm. This requires either:

- The DIAMETER realm resolves to a `NAPTR` record which returns an `AAA+D2T` entry which in turn resolves to `SRV`/`A` records which finally resolve to IPv4 addresses of reachable nodes in the realm.
- The DIAMETER realm resolves to a collection of `A` records which directly resolve to reachable nodes in the realm.

Configuring the billing realm

To point Ralf at the billing DIAMETER realm, add the following line to `/etc/clearwater/shared_config` and follow this process to apply the change

```
billing_realm=<DIAMETER billing realm>
```

Selecting a specific CDF in the realm

Note: Bono only has support for selecting CDF identities based of static configuration of a single identity. Other P-CSCFs may have support for load-balancing or enabling backup CDF identities.

If you have a CDF set up to receive Rf billing messages from your deployment, you will need to modify the `/etc/clearwater/shared_config` file and follow this process to apply the change:

```
cdf_identity=<CDF DIAMETER Identity>
```

Restrictions

The very first release of Ralf, from the Counter-Strike release of Project Clearwater, does not generate Rf billing messages since the related changes to Sprout and Bono (to report billable events) were not enabled. This version was released to allow systems integrators to get a head start on spinning up and configuring Ralf nodes rather than having to wait for the next release.

Clearwater IP Port Usage

The nodes in Clearwater attempt talk to each other over IP. This document lists the ports that are used by a deployment of Clearwater.

All nodes

All nodes need to allow the following ICMP messages:

```
Echo Request
Echo Reply
Destination Unreachable
```

They also need the following ports open to the world (`0.0.0.0/0`):

```
TCP/22 for SSH access
```

If your deployment uses SNMP monitoring software (cacti for example), each node will also have to open appropriate ports for this service.

- SNMP

  ```
  UDP/161-162
  ```

All nodes also need the following ports open to all other nodes for automatic clustering and configuration sharing:

- etcd

```
TCP/2380
TCP/4000
```

All-in-one

All-in-one nodes need the following ports opened to the world

- Web UI

```
TCP/80
TCP/443
```

- STUN signaling:

```
TCP/3478
UDP/3478
```

- SIP signaling:

```
TCP/5060
UDP/5060
TCP/5062
```

- RTP forwarding:

```
UDP/32768-65535
```

Ellis

The Ellis node needs the following ports opened to the world:

- Web UI

```
TCP/80
TCP/443
```

Bono

The Bono nodes need the following ports opened to the world:

- STUN signaling:

```
TCP/3478
UDP/3478
```

- SIP signaling:

```
TCP/5060
UDP/5060
TCP/5062
```

- RTP forwarding:

```
UDP/32768-65535
```

They also need the following ports open to all other Bono nodes and to all the Sprout nodes:

- Internal SIP signaling:

```
TCP/5058
```

Sprout

The Sprout nodes need the following ports open to all Bono nodes:

- Internal SIP signaling:

```
TCP/5054
TCP/5052
```

They also need the following ports opened to all Vellum nodes:

- Chronos:

```
TCP/9888
```

They also need the following ports opened to all Dime nodes:

- Registration Termination Requests (if using an HSS):

```
TCP/9888
```

They also need the following ports opened to the world:

- HTTP interface (if including a Memento AS):

```
TCP/443
```

Dime

The Dime nodes need the following ports open to all the Sprout nodes and the Ellis node:

- RESTful interface:

```
TCP/8888
```

They also need the following ports open to just the Ellis node:

- RESTful interface:

```
TCP/8889
```

They also need the following ports open to all the Sprout, Bono and Vellum nodes:

- RESTful interface:

```
TCP/10888
```

Homer

The Homer nodes need the following ports open to all the Sprout nodes and the Ellis node:

- RESTful interface:

```
TCP/7888
```

Vellum

The Vellum nodes need the following ports open to all other Vellum nodes:

- Chronos:

```
TCP/7253
```

- Memcached:

```
TCP/11211
```

- Cassandra:

```
TCP/7000
```

They also need the following ports open to all Sprout and Dime nodes:

- Chronos:

```
TCP/7253
```

- Astaire:

```
TCP/11311
```

They also need the following ports open to all Homer and Dime nodes (and all Sprout nodes, if including a Memento AS):

- Cassandra:

```
TCP/9160
```

Standalone Application Servers

Standalone Project Clearwater application servers (e.g. Memento and Gemini) need the following ports open to all Sprout nodes:

- SIP signaling:

```
TCP/5054
```

They also need the following ports opened to the world (if they include a Memento AS):

- HTTP interface:

```
TCP/443
```

Clearwater DNS Usage

DNS is the Domain Name System. It maps service names to hostnames and then hostnames to IP addresses. Clearwater uses DNS.

This document describes

- Clearwater's DNS strategy and requirements
- how to configure AWS Route 53 and BIND to meet these.

DNS is also used as part of the ENUM system for mapping E.164 numbers to SIP URIs. This isn't discussed in this document - instead see the separate ENUM document.

If you are installing an All-in-One Clearwater node, you do not need any DNS records and can ignore the rest of this page.

Strategy

Clearwater makes heavy use of DNS to refer to its nodes. It uses it for

- identifying individual nodes, e.g. sprout-1.example.com might resolve the IP address of the first sprout node
- identifying the nodes within a cluster, e.g. sprout.example.com resolves to all the IP addresses in the cluster
- fault-tolerance

Clearwater also supports using DNS for identifying non-Clearwater nodes. In particular, it supports DNS for identifying SIP peers using NAPTR and SRV records, as described in RFC 3263.

Resiliency

By default, Clearwater routes all DNS requests through an instance of dnsmasq running on localhost. This round-robins requests between the servers in /etc/resolv.conf, as described in its FAQ:

> By default, dnsmasq treats all the nameservers it knows about as equal: it picks the one to use using an algorithm designed to avoid nameservers which aren't responding.

If the `signaling_dns_server` option is set in `/etc/clearwater/shared_config` (which is mandatory when using traffic separation), Clearwater will not use dnsmasq. Instead, resiliency is achieved by being able to specify up to three servers in a comma-separated list (e.g. `signaling_dns_server=1.2.3.4,10.0.0.1,192.168.1.1`), and Clearwater will fail over between them as follows:

- It will always query the first server in the list first
- If this returns SERVFAIL or times out (which happens after a randomised 500ms-1000ms period), it will resend the query to the second server
- If this returns SERVFAIL or times out, it will resend the query to the third server
- If all servers return SERVFAIL or time out, the DNS query will fail

Clearwater caches DNS responses for several minutes (to reduce the load on DNS servers, and the latency introduced by querying them). If a cache entry is stale, but the DNS servers return SERVFAIL or time out when Clearwater attempts to refresh it, Clearwater will continue to use the cached value until the DNS servers become responsive again. This minimises the impact of a DNS server failure on calls.

Requirements

DNS Server

Clearwater requires the DNS server to support

- RFC 1034 and RFC 1035 - basic DNS
- RFC 2181 - clarifications to DNS
- RFC 2782 - SRV records
- RFC 3596 - AAAA records (if IPv6 is required).

Support for latency-based routing and health-checking are required for multi-site deployments.

Support for RFC 2915 (NAPTR records) is also suggested, but not required. NAPTR records specify the transport (UDP, TCP, etc.) to use for a particular service - without it, UEs will default (probably to UDP).

AWS Route 53 supports all these features except NAPTR. BIND supports all these features except latency-based routing (although there is a patch for this) and health-checking.

DNS Records

Clearwater requires the following DNS records to be configured.

- Bono
  - `bono-1.<zone>`, `bono-2.<zone>`... (A and/or AAAA) - per-node records for bono
  - `<zone>` (A and/or AAAA) - cluster record for bono, resolving to all bono nodes - used by UEs that don't support RFC 3263 (NAPTR/SRV)
  - `<zone>` (NAPTR, optional) - specifies transport requirements for accessing bono - service `SIP+D2T` maps to `_sip._tcp.<zone>` and `SIP+D2U` maps to `_sip._udp.<zone>`
  - `_sip._tcp.<zone>` and `_sip._udp.<zone>` (SRV) - cluster SRV records for bono, resolving to port 5060 for all of the per-node records
- Sprout
  - `sprout-1.<zone>`, `sprout-2.<zone>`... (A and/or AAAA) - per-node records for sprout
  - `scscf.sprout.<zone>` (A and/or AAAA) - cluster record for sprout, resolving to all sprout nodes that provide S-CSCF function - used by P-CSCFs that don't support RFC 3263 (NAPTR/SRV)
  - `scscf.sprout.<zone>` (NAPTR, optional) - specifies transport requirements for accessing sprout - service `SIP+D2T` maps to `_sip._tcp.scscf.sprout.<zone>`
  - `_sip._tcp.scscf.sprout.<zone>` (SRV) - cluster SRV record for sprout, resolving to port 5054 for all of the per-node records
  - `icscf.sprout.<zone>` (A and/or AAAA) - cluster record for sprout, resolving to all sprout nodes that provide I-CSCF function - used by P-CSCFs that don't support RFC 3263 (NAPTR/SRV)
  - `icscf.sprout.<zone>` (NAPTR, optional) - specifies transport requirements for accessing sprout - service `SIP+D2T` maps to `_sip._tcp.icscf.sprout.<zone>`
  - `_sip._tcp.icscf.sprout.<zone>` (SRV) - cluster SRV record for sprout, resolving to port 5052 for all of the per-node records
- Dime
  - `dime-1.<zone>`, `dime-2.<zone>`... (A and/or AAAA) - per-node records for dime

  - `hs.<zone>` (A and/or AAAA) - cluster record for homestead, resolving to all dime nodes
  - `ralf.<zone>` (A and/or AAAA) - cluster record for ralf, resolving to all dime nodes
- Homer
  - `homer-1.<zone>`, `homer-2.<zone>`... (A and/or AAAA) - per-node records for homer
  - `homer.<zone>` (A and/or AAAA) - cluster record for homer, resolving to all homer nodes
- Vellum
  - `vellum-1.<zone>`, `vellum-2.<zone>`... (A and/or AAAA) - per-node records for vellum
  - `vellum.<zone>` (A and/or AAAA) - cluster record for vellum, resolving to all vellum nodes
- Ellis
  - `ellis-1.<zone>` (A and/or AAAA) - per-node record for ellis
  - `ellis.<zone>` (A and/or AAAA) - "cluster"/access record for ellis
- Standalone application server (e.g. gemini/memento)
  - `<standalone name>-1.<zone>` (A and/or AAAA) - per-node record for each standalone application server
  - `<standalone name>.<zone>` (A and/or AAAA) - "cluster"/access record for the standalone application servers

Of these, the following must be resolvable by UEs - the others need only be resolvable within the core of the network. If you have a NAT-ed network, the following must resolve to public IP addresses, while the others should resolve to private IP addresses.

- Bono
  - `<zone>` (A and/or AAAA)
  - `<zone>` (NAPTR, optional)
  - `_sip._tcp.<zone>` and `_sip._udp.<zone>` (SRV)
- Ellis
  - `ellis.<zone>` (A and/or AAAA)
- Memento
  - `memento.<zone>` (A and/or AAAA)

If you are not deploying with some of these components, you do not need the DNS records to be configured for them. For example, if you are using a different P-CSCF (and so don't need bono), you don't need the bono DNS records. Likewise, if you are deploying with an external HSS (and so don't need ellis), you don't need the ellis DNS records.

If your deployment is geographically redundant, then you need a DNS record per site for every cluster record mentioned above. For example, in a GR deployment with two sites, siteA and siteB, the requirements for Dime are:

```
*   `dime-1.<zone>`, `dime-2.<zone>`... (A and/or AAAA) - per-node records for Dime␣
↪(one record for each node in each site)
*   `hs.siteA.<zone>` (A and/or AAAA) - cluster record for Homestead, resolving to␣
↪all Dime nodes in siteA.
*   `hs.siteB.<zone>` (A and/or AAAA) - cluster record for Homestead, resolving to␣
↪all Dime nodes in siteB.
*   `ralf.siteA.<zone>` (A and/or AAAA) - cluster record for Ralf, resolving to all␣
↪Dime nodes in siteA.
*   `ralf.siteB.<zone>` (A and/or AAAA) - cluster record for Ralf, resolving to all␣
↪Dime nodes in siteB.
```

The exceptions to the above are Bono and Ellis.

Ellis doesn't support geographic redundancy (or even there being more than one Ellis), so there's no need to have multiple DNS records.

Bono needs to be able to contact the Sprout nodes in each site, so it needs to have a DNS record that can resolve to all Sprouts; the expected Sprout/Bono DNS records for a GR deployment with two sites, siteA and siteB, are described below (this only includes the S-CSCF records for simplicity).

```
*   `bono-1.<zone>`, `bono-2.<zone>`... (A and/or AAAA) - per-node records for Bono␣
↪(one record for each node in each site)
*   `<zone>` (A and/or AAAA) - cluster record for Bono, resolving to all bono nodes␣
↪in all sites - used by UEs that don't support RFC 3263 (NAPTR/SRV)
*   `<zone>` (NAPTR, optional) - specifies transport requirements for accessing Bono -
↪ service `SIP+D2T` maps to `_sip._tcp.<zone>` and `SIP+D2U` maps to `_sip._udp.
↪<zone>`
*   `_sip._tcp.<zone>` and `_sip._udp.<zone>` (SRV) - cluster SRV records for Bono,␣
↪resolving to port 5060 for all of the per-node records
*   `sprout-1.<zone>`, `sprout-2.<zone>`... (A and/or AAAA) - per-node records for␣
↪Sprout (one record for each node in each site)
*   `scscf.sprout.<zone>` (A and/or AAAA) - cluster record for Sprout, resolving to␣
↪all Sprout nodes in all sites that provide S-CSCF function - used by P-CSCFs that␣
↪don't support RFC 3263 (NAPTR/SRV)
*   `scscf.sprout.<zone>` (NAPTR, optional) - specifies transport requirements for␣
↪accessing Sprout - service `SIP+D2T` maps to `_sip._tcp.scscf.sprout.<zone>`
*   `_sip._tcp.scscf.sprout.<zone>` (SRV) - cluster SRV record for Sprout, resolving␣
↪to port 5054 for all of the per-node records
*   `scscf.sprout.siteA.<zone>` (A and/or AAAA) - cluster record for Sprout,␣
↪resolving to all Sprout nodes in siteA that provide S-CSCF function - used by P-
↪CSCFs that don't support RFC 3263 (NAPTR/SRV)
*   `scscf.sprout.siteA.<zone>` (NAPTR, optional) - specifies transport requirements␣
↪for accessing Sprout - service `SIP+D2T` maps to `_sip._tcp.scscf.sprout.siteA.
↪<zone>`
*   `_sip._tcp.scscf.sprout.siteA.<zone>` (SRV) - cluster SRV record for Sprout,␣
↪resolving to port 5054 for all of the per-node records in siteA
*   `scscf.sprout.siteB.<zone>` (A and/or AAAA) - cluster record for Sprout,␣
↪resolving to all Sprout nodes in siteB that provide S-CSCF function - used by P-
↪CSCFs that don't support RFC 3263 (NAPTR/SRV)
*   `scscf.sprout.siteB.<zone>` (NAPTR, optional) - specifies transport requirements␣
↪for accessing Sprout - service `SIP+D2T` maps to `_sip._tcp.scscf.sprout.siteB.
↪<zone>`
*   `_sip._tcp.scscf.sprout.siteB.<zone>` (SRV) - cluster SRV record for Sprout,␣
↪resolving to port 5054 for all of the per-node records in siteB
```

Configuration

Clearwater can work with any DNS server that meets the *requirements above*. However, most of our testing has been performed with

- AWS Route 53 - see *configuration instructions*
- BIND - see *configuration instructions*.

The Clearwater nodes also need to know the identity of their DNS server. Ideally, this is done via DHCP within your virtualization infrastructure. Alternatively, you can *configure it manually*.

The UEs need to know the identity of the DNS server too. In a testing environment, you may be able to use DHCP or manual configuration. In a public network, you will need to register the <zone> domain name you are using and arranging for an NS record for <zone> to point to your DNS server.

AWS Route 53

Clearwater's automated install automatically configures AWS Route 53. There is no need to follow the following instructions if you are using the automated install.

The official AWS Route 53 documentation is a good reference, and most of the following steps are links into it.

To use AWS Route 53 for Clearwater, you need to

- create a domain
- create record sets

Note that AWS Route 53 does not support NAPTR records.

BIND

To use BIND, you need to

- install it
- create an entry for your "zone" (DNS suffix your deployment uses)
- configure the zone with a "zone file"
- restart BIND.

Note that BIND does not support latency-based routing or health-checking.

Installation

To install BIND on Ubuntu, issue `sudo apt-get install bind9`.

Creating Zone Entry

To create an entry for your zone, edit the `/etc/bind/named.conf.local` file to add a line of the following form, replacing <zone> with your zone name.

```
zone "<zone>" IN { type master; file "/etc/bind/db.<zone>"; };
```

Configuring Zone

Zones are configured through "zone files" (defined in RFC 1034 and RFC 1035).

If you followed the instructions above, the zone file for your zone is at `/etc/bind/db.<zone>`.

For Clearwater, you should be able to adapt the following example zone file by correcting the IP addresses and duplicating (or removing) entries where you have more (or fewer) than 2 nodes in each tier.

```
$TTL 5m ; Default TTL

; SOA, NS and A record for DNS server itself
@                 3600 IN SOA  ns admin ( 2014010800 ; Serial
                                          3600       ; Refresh
                                          3600       ; Retry
                                          3600       ; Expire
                                          300 )      ; Minimum TTL
@                 3600 IN NS   ns
ns                3600 IN A    1.0.0.1 ; IPv4 address of BIND server
ns                3600 IN AAAA 1::1    ; IPv6 address of BIND server

; bono
; ====
;
; Per-node records - not required to have both IPv4 and IPv6 records
bono-1                IN A     2.0.0.1
bono-2                IN A     2.0.0.2
bono-1                IN AAAA  2::1
bono-2                IN AAAA  2::2
;
; Cluster A and AAAA records - UEs that don't support RFC 3263 will simply
; resolve the A or AAAA records and pick randomly from this set of addresses.
@                     IN A     2.0.0.1
@                     IN A     2.0.0.2
@                     IN AAAA  2::1
@                     IN AAAA  2::2
;
; NAPTR and SRV records - these indicate a preference for TCP and then resolve
; to port 5060 on the per-node records defined above.
@                     IN NAPTR 1 1 "S" "SIP+D2T" "" _sip._tcp
@                     IN NAPTR 2 1 "S" "SIP+D2U" "" _sip._udp
_sip._tcp             IN SRV   0 0 5060 bono-1
_sip._tcp             IN SRV   0 0 5060 bono-2
_sip._udp             IN SRV   0 0 5060 bono-1
_sip._udp             IN SRV   0 0 5060 bono-2

; sprout
; ======
;
; Per-node records - not required to have both IPv4 and IPv6 records
sprout-1              IN A     3.0.0.1
sprout-2              IN A     3.0.0.2
sprout-1              IN AAAA  3::1
sprout-2              IN AAAA  3::2
;
; Cluster A and AAAA records - P-CSCFs that don't support RFC 3263 will simply
; resolve the A or AAAA records and pick randomly from this set of addresses.
sprout                IN A     3.0.0.1
sprout                IN A     3.0.0.2
sprout                IN AAAA  3::1
sprout                IN AAAA  3::2
;
; Cluster A and AAAA records - P-CSCFs that don't support RFC 3263 will simply
; resolve the A or AAAA records and pick randomly from this set of addresses.
scscf.sprout          IN A     3.0.0.1
scscf.sprout          IN A     3.0.0.2
scscf.sprout          IN AAAA  3::1
```

```
scscf.sprout           IN AAAA  3::2
;
; NAPTR and SRV records - these indicate TCP support only and then resolve
; to port 5054 on the per-node records defined above.
sprout                 IN NAPTR 1 1 "S" "SIP+D2T" "" _sip._tcp.sprout
_sip._tcp.sprout       IN SRV   0 0 5054 sprout-1
_sip._tcp.sprout       IN SRV   0 0 5054 sprout-2
;
; NAPTR and SRV records for S-CSCF - these indicate TCP support only and
; then resolve to port 5054 on the per-node records defined above.
scscf.sprout           IN NAPTR 1 1 "S" "SIP+D2T" "" _sip._tcp.scscf.sprout
_sip._tcp.scscf.sprout IN SRV   0 0 5054 sprout-1
_sip._tcp.scscf.sprout IN SRV   0 0 5054 sprout-2
;
; Cluster A and AAAA records - P-CSCFs that don't support RFC 3263 will simply
; resolve the A or AAAA records and pick randomly from this set of addresses.
icscf.sprout           IN A     3.0.0.1
icscf.sprout           IN A     3.0.0.2
icscf.sprout           IN AAAA  3::1
icscf.sprout           IN AAAA  3::2
;
; NAPTR and SRV records for I-CSCF - these indicate TCP support only and
; then resolve to port 5052 on the per-node records defined above.
icscf.sprout           IN NAPTR 1 1 "S" "SIP+D2T" "" _sip._tcp.icscf.sprout
_sip._tcp.icscf.sprout IN SRV   0 0 5052 sprout-1
_sip._tcp.icscf.sprout IN SRV   0 0 5052 sprout-2

; dime
; =========
;
; Per-node records - not required to have both IPv4 and IPv6 records
dime-1                 IN A     4.0.0.1
dime-2                 IN A     4.0.0.2
dime-1                 IN AAAA  4::1
dime-2                 IN AAAA  4::2
;
; Cluster A and AAAA records - sprout, bono and ellis pick randomly from these.
hs                     IN A     4.0.0.1
hs                     IN A     4.0.0.2
hs                     IN AAAA  4::1
hs                     IN AAAA  4::2
ralf                   IN A     4.0.0.1
ralf                   IN A     4.0.0.2
ralf                   IN AAAA  4::1
ralf                   IN AAAA  4::2
;
; (No need for NAPTR or SRV records as dime doesn't handle SIP traffic.)

; homer
; =====
;
; Per-node records - not required to have both IPv4 and IPv6 records
homer-1                IN A     5.0.0.1
homer-2                IN A     5.0.0.2
homer-1                IN AAAA  5::1
homer-2                IN AAAA  5::2
;
; Cluster A and AAAA records - sprout picks randomly from these.
```

```
homer                   IN A     5.0.0.1
homer                   IN A     5.0.0.2
homer                   IN AAAA  5::1
homer                   IN AAAA  5::2
;
; (No need for NAPTR or SRV records as homer doesn't handle SIP traffic.)

; vellum
; =====
;
; Per-node records - not required to have both IPv4 and IPv6 records
vellum-1                IN A     6.0.0.1
vellum-2                IN A     6.0.0.2
vellum-1                IN AAAA  6::1
vellum-2                IN AAAA  6::2
;
; Cluster A and AAAA records - sprout, homer and dime pick randomly from these.
vellum                  IN A     6.0.0.1
vellum                  IN A     6.0.0.2
vellum                  IN AAAA  6::1
vellum                  IN AAAA  6::2
;
; (No need for NAPTR or SRV records as vellum doesn't handle SIP traffic.)

; ellis
; =====
;
; ellis is not clustered, so there's only ever one node.
;
; Per-node record - not required to have both IPv4 and IPv6 records
ellis-1                 IN A     7.0.0.1
ellis-1                 IN AAAA  7::1
;
; "Cluster"/access A and AAAA record
ellis                   IN A     7.0.0.1
ellis                   IN AAAA  7::1
```

Restarting

To restart BIND, issue `sudo service bind9 restart`. Check /var/log/syslog for any error messages.

Client Configuration

Clearwater nodes need to know the identity of their DNS server. Ideally, this is achieved through DHCP. There are two main situations in which it might need to be configured manually.

- When DNS configuration is not provided via DHCP.
- When incorrect DNS configuration is provided via DHCP.

Either way, you must

- create an `/etc/dnsmasq.resolv.conf` file containing the desired DNS configuration (probably just the single line `nameserver <IP address>`)
- add `RESOLV_CONF=/etc/dnsmasq.resolv.conf` to `/etc/default/dnsmasq`

- run `service dnsmasq restart`.

(As background, dnsmasq is a DNS forwarder that runs on each Clearwater node to act as a cache. Local processes look in `/etc/resolv.conf` for DNS configuration, and this points them to localhost, where dnsmasq runs. In turn, dnsmasq takes its configuration from `/etc/dnsmasq.resolv.conf`. By default, dnsmasq would use `/var/run/dnsmasq/resolv.conf`, but this is controlled by DHCP.)

IPv6 AAAA DNS lookups

Clearwater can be installed on an IPv4-only system, an IPv6-only system, or a system with both IPv4 and IPv6 addresses (though the Clearwater software does not use both IPv4 and IPv6 at the same time).

Normally, systems with both IPv4 and IPv6 addresses will prefer IPv6, performing AAAA lookups first and only trying an A record lookup if that fails. This may cause problems (or be inefficient) if you know that all your Clearwater DNS records are A records.

In this case, you can configure a preference for A lookups by editing `/etc/gai.conf` and commenting out the line `precedence ::ffff:0:0/96 100` (as described at http://askubuntu.com/questions/32298/prefer-a-ipv4-dns-lookups-before-aaaaipv6-lookups).

ENUM Guide

ENUM is a system for mapping PSTN numbers to SIP URIs using DNS NAPTR records, and which sprout supports. This article describes

- briefly, what ENUM is
- what we support
- how to configure a server to respond to ENUM queries.

ENUM overview

The NAPTR article on Wikipedia gives a pretty good overview of how ENUM works. Essentially, you

- convert your PSTN number into a domain name by stripping all punctuation, reversing the order of the digits, separating them with dots and suffixing with "e164.arpa" (e.g. 12012031234 becomes 4.3.2.1.3.0.2.1.0.2.1.e164.arpa)
- issue a DNS NAPTR query for this domain and receive the response (which is for the best prefix match to your number)
- parse the response, which contains a series of regular expressions and replacements
- find the best match for your number (after stripping all punctuation except any leading +) and apply the replacement
- if the NAPTR response indicated this match was "terminal", then use the resulting SIP URI - if not, use the output of this pass as input for another ENUM query.

The RFCs to refer to for a definitive position are RFC 3401 (which covers Dynamic Delegation Discovery System (DDDS), on which ENUM is built) and RFC 3761 (which covers ENUM itself). Also relevant is RFC 4769 (which specifies ENUM service types).

Clearwater ENUM Support

Clearwater supports ENUM as set out in RFC 3761 and RFC 4769, with the following points/omissions.

- ENUM processing should only be applied to TEL URIs and SIP URIs that specify that this was dialed by the user. Since no SIP UAs that we're aware of indicate that the SIP URI was dialed by a user, we assume that any all-numeric (including +) URIs should have ENUM processing applied.
- RFC 3761 section 2.4 says that the ENUM replacement string is UTF-8-encoded. We do not support this - all characters must be ASCII (i.e. a subset of UTF-8). It is not particularly meaningful to use UTF-8 replacements for SIP URIs as SIP URIs themselves must be ASCII.
- RFC 3761 section 6.1 says that a deployed ENUM service SHOULD include mechanisms to ameliorate security threats and mentions using DNSSEC. We don't support DNSSEC, so other security approaches (such as private ENUM servers) must be used.
- RFC 4769 describes ENUM rules that output TEL URIs. Since we have no way (apart from ENUM itself) of routing to TEL URIs, we ignore such rules.
- RFC 3761 section 2.1 describes the construction of the Application Unique String from the E.164 number. We have no way of converting a user-dialed number into an E.164 number, so we assume that the number provided by the user is E.164.

Deciding on ENUM rules

ENUM rules will vary with your deployment, but the (JSON-ized) rules from an example deployment might form a reasonable basis:

```
{
    "number_blocks" : [
        {   "name" : "Demo system number 2125550270",
            "prefix" : "2125550270",
            "regex" : "!(^.*$)!sip:\\1@10.147.226.2!"
        },
        {
            "name" : "Clearwater external numbers 2125550271-9",
            "prefix" : "212555027",
            "regex" : "!(^.*$)!sip:+1\\1@ngv.example.com!"
        },
        {
            "name" : "Clearwater external numbers +12125550271-9",
            "prefix" : "+1212555027",
            "regex" : "!(^.*$)!sip:\\1@ngv.example.com!"
        },
        {
            "name" : "Clearwater external number 2125550280",
            "prefix" : "2125550280",
            "regex" : "!(^.*$)!sip:+1\\1@ngv.example.com!"
        },
        {
            "name" : "Clearwater external number +12125550280",
            "prefix" : "+12125550280",
            "regex" : "!(^.*$)!sip:\\1@ngv.example.com!"
        },
        {   "name" : "Clearwater internal numbers",
            "prefix" : "650555",
            "regex" : "!(^.*$)!sip:\\1@ngv.example.com!"
        },
```

```
        {   "name" : "Clearwater internal numbers dialled with +1 prefix",
            "prefix" : "+1650555",
            "regex" : "!^+1(.*$)!sip:\\1@ngv.example.com!"
        },
        {   "name" : "NANP => SIP trunk",
            "prefix" : "",
            "regex" : "!(^.*$)!sip:\\1@10.147.226.2!"
        }
    ]
}
```

Configuring ENUM rules

Normally, the ENUM server would be an existing part of the customer's network (not part of Clearwater itself) but, for testing and demonstration, it is useful to be able to configure our own.

Unfortunately, AWS Route 53 does not support the required NAPTR records, so we can't use this. Fortunately, BIND (easy to install) does, and dnsmasq (our standard DNS forwarder on Clearwater nodes) does to a limited extent. This section describes how to configure BIND or dnsmasq to respond to ENUM queries.

You can set up ENUM rules on either BIND or dnsmasq. BIND is the better choice, but requires installing an additional package and a few additional file tweaks. dnsmasq is the weaker choice - it does not support wildcard domains properly, so rules for each directory number must be added separately.

BIND

Create a new node to run BIND, and open port 53 on it to the world (`0.0.0.0/0`).

If you are using chef, you can do this by adding a new 'enum' node.

On the new node,

1. install bind by typing "sudo apt-get install bind9"
2. modify /etc/bind/named.conf to add a line 'include "/etc/bind/named.conf.e164.arpa";' - we'll create that file next
3. create /etc/bind/named.conf.e164.arpa, with the following contents - we'll create the /etc/bind/e164.arpa.db file next

```
zone "e164.arpa" {
        type master;
        file "/etc/bind/e164.arpa.db";
};
```

4. create /etc/bind/e164.arpa.db, with the following header

```
$TTL 1h
@ IN SOA e164.arpa ngv-admin@example.com (
                                                    2009010910 ;serial
                                                    3600 ;refresh
                                                    3600 ;retry
                                                    3600 ;expire
                                                    3600 ;minimum TTL
)
@ IN NS e164.arpa.
@ IN A <this server's IP address>
```

5. add additional rules of the form '<enum domain name> <order> <preference> "<flags>" "<service>" "<regexp>" .' to this file (being careful to maintain double-quotes and full-stops)
   - enum domain name is the domain name for which you want to return this regular expression - for example, if you want to use a regular expression for number "1201", this would be "1.0.2.1.e164.arpa" - if you want to specify a wildcard domain (i.e. a prefix match), use *.1.0.2.1.e164.arpa - note, however, that non-wildcard children of a wildcard domain are not themselves wildcarded, e.g. if you have a domain *.1.e164.arpa and a domain 2.1.e164.arpa, a query for 3.2.1.e164.arpa would not resolve (even though it matches *.1.e164.arpa)
   - order specifies the order in which this rule is applied compared to others - rules of order 1 and processed before those of order 2 (and rules of order 2 are not processed if one of the matching rules of of order 1 are non-terminal)
   - preference specifies how preferable one rule is compared to another - if both rules match, the one with the higher preference is used
   - flags specifies how a match should be processed - it can either be blank (meaning apply the regular expression and then continue) or "U" (meaning the rule is terminal and, after applying the regular expression, no further rules should be processed)
   - service must be "E2U+SIP", indicating ENUM rather than other services
   - regexp must be a regular expression to apply and is of the form !<pattern>!<replacement>! - note that the "!" delimiter is only by convention and can be replaced with other symbols (such as "/") if "!" occurs within the pattern or replacement.
6. restart bind by typing "sudo service bind9 restart"
7. check /var/log/syslog for errors reported by bind on start-up
8. test your configuration by typing "dig -t NAPTR <enum domain name>" and checking you get back the responses you just added.

As an example, the JSON-ized ENUM rules for an example system (above), translate to the following entries in /etc/bind/e164.arpa.db.

```
; Demo system number 2125550270
0.7.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .

; Demo system number +12125550270
0.7.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .

; Clearwater external numbers 2125550271-9
*.7.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:+1\\1@ngv.example.com!" .

; Clearwater external numbers +12125550271-9
; Note that we can't define a domain name containing + so we must define a
; domain without it and then match a telephone number starting with a + (if
; present) or (if not) use the default route via the SIP trunk.
*.7.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(\+^.*$)!sip:\\1@ngv.example.com!"␣
↪.
*.7.2.0.5.5.5.2.1.2.1 IN NAPTR 2 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .

; Clearwater external number 2125550280
0.8.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:+1\\1@ngv.example.com!" .

; Clearwater external number +12125550280
; Note that we can't define a domain name containing + so we must define a
; domain without it and then match a telephone number starting with a + (if
; present) or (if not) use the default route via the SIP trunk.
```

```
0.8.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(\+^.*$)!sip:\\1@ngv.example.com!"
↪.
0.8.2.0.5.5.5.2.1.2.1 IN NAPTR 2 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .

; Clearwater internal numbers
*.5.5.5.0.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@ngv.example.com!" .

; Clearwater internal numbers dialled with +1 prefix
; Note that we can't define a domain name containing + so we must define a
; domain without it and then match a telephone number starting with a + (if
; present) or (if not) use the default route via the SIP trunk.
*.5.5.5.0.5.6.1 IN NAPTR 1 1 "u" "E2U+sip" "!^\+1(.*$)!sip:\\1@ngv.example.com!" .
*.5.5.5.0.5.6.1 IN NAPTR 2 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .

; NANP => SIP trunk.
* IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
7.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
7.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
8.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.8.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
8.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.8.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.0.8.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
```

```
*.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.0.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.0.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.5.0.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
*.5.5.0.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
5.5.5.0.5.6 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
```

dnsmasq

Before configuring dnsmasq, you need to find a suitable host.

- For a deployment with only one sprout node you should just be able to use the sprout node itself.
- For deployments with more than one sprout node, you'll need to either create a new node or pick one of your sprout nodes (bearing in mind that it will become a single point-of-failure) and then configure dnsmasq on all sprouts to query this server for enum lookups by creating an "/etc/dnsmasq.d/enum-forwarding" file containing a single line of the form "server=/e164.arpa/<DNS IP address/" and then restarting dnsmasq with "sudo service dnsmasq restart". Note that you'll also need to open UDP port 53 (DNS) to this server in the security group.

On the host you've chosen,

1. ensure dnsmasq is installed (it is standard on all Clearwater nodes) by typing "sudo apt-get install dnsmasq"
2. create an "/etc/dnsmasq.d/enum" file containing lines of the form "naptr-record=<enum domain name>,<order>,<preference>,<flags>,<service>,<regexp>" where
   - enum domain name is the domain name for which you want to return this regular expression - for example, if you want to use a regular expression for number "1201", this would be "1.0.2.1.e164.arpa"
   - order specifies the order in which this rule is applied compared to others - rules of order 1 and processed before those of order 2 (and rules of order 2 are not processed if one of the matching rules of of order 1 are non-terminal)
   - preference specifies how preferable one rule is compared to another - if both rules match, the one with the higher preference is used
   - flags specifies how a match should be processed - it can either be blank (meaning apply the regular expression and then continue) or "U" (meaning the rule is terminal and, after applying the regular expression, no further rules should be processed)
   - service must be "E2U+SIP", indicating ENUM rather than other services
   - regexp must be a regular expression to apply and is of the form !<pattern>!<replacement>! - note that the "!" delimeter is only by convention and can be replaced with other symbols (such as "/") if "!" occurs within the pattern or replacement.
3. restart dnsmasq by typing "sudo service dnsmasq restart"
4. test your configuration by typing "dig -t NAPTR <enum domain name>" and checking you get back the responses you just added.

As an example, the JSON-ized ENUM rules for an example system (above), translate to the following entries in /etc/dnsmasq.d/enum.

```
# Demo system number 2125550270
naptr-record=0.7.2.0.5.5.5.2.1.2.e164.arpa,1,1,U,E2U+SIP,!(^.*$)!sip:\1@10.147.226.2!

# Clearwater external numbers 2125550271-9
```

```
naptr-record=7.2.0.5.5.5.2.1.2.e164.arpa,1,1,U,E2U+SIP,!(^.*$)!sip:+1\1@ngv.example.
↪com!

# Clearwater external numbers +12125550271-9
# Note that we can't define a domain name containing + so we must define a # domain␣
↪without it and then match a telephone number starting with a + (if
# present) or (if not) use the default route via the SIP trunk.
naptr-record=7.2.0.5.5.5.2.1.2.1.e164.arpa,1,1,U,E2U+SIP,!(\+^.*$)!sip:\1@ngv.example.
↪com!
naptr-record=7.2.0.5.5.5.2.1.2.1.e164.arpa,2,1,U,E2U+SIP,!(^.*$)!sip:\1@10.147.226.2!

# Clearwater external number 2125550280
naptr-record=0.8.2.0.5.5.5.2.1.2.e164.arpa,1,1,U,E2U+SIP,!(^.*$)!sip:+1\1@ngv.example.
↪com!

# Clearwater external number +12125550280
# Note that we can't define a domain name containing + so we must define a
# domain without it and then match a telephone number starting with a + (if
# present) or (if not) use the default route via the SIP trunk.
naptr-record=0.8.2.0.5.5.5.2.1.2.1.e164.arpa,1,1,U,E2U+SIP,!(\+^.*$)!sip:\1@ngv.
↪example.com!
naptr-record=0.8.2.0.5.5.5.2.1.2.1.e164.arpa,2,1,U,E2U+SIP,!(^.*$)!sip:\1@10.147.226.
↪2!

# Clearwater internal numbers
naptr-record=5.5.5.0.5.6.e164.arpa,1,1,U,E2U+SIP,!(^.*$)!sip:\1@ngv.example.com!

# Clearwater internal numbers dialled with +1 prefix
# Note that we can't define a domain name containing + so we must define a
# domain without it and then match a telephone number starting with a + (if
# present) or (if not) use the default route via the SIP trunk.
naptr-record=5.5.5.0.5.6.1.e164.arpa,1,1,U,E2U+SIP,!^\+1(.*$)!sip:\1@ngv.example.com!
naptr-record=5.5.5.0.5.6.1.e164.arpa,2,1,U,E2U+SIP,!(^.*$)!sip:\1@10.147.226.2!

# NANP => SIP trunk
naptr-record=e164.arpa,1,1,U,E2U+SIP,!(^.*$)!sip:\1@10.147.226.2!
```

ENUM Domain Suffix

RFC 3761 mandates that domain names used during ENUM processing are suffixed with .e164.arpa. Obviously, this means that there can only be one such public domain. If you need your domain to be public (rather than private as set up above), you can instead change the suffix, e.g. to .e164.arpa.ngv.example.com, by

- configuring sprout to use this suffix via the –enum-suffix parameter
- configuring your DNS server to respond to this domain rather than .e164.arpa (by search-replacing in the config files described above)
- configuring Route 53 to forward DNS requests for e164.arpa.ngv.example.com to your DNS server by creating an NS (Name Server) record with name "e164.arpa.ngv.example.com" and value set to the name/IP address of your DNS server.

ENUM and Sprout

To enable ENUM lookups on Sprout, edit `/etc/clearwater/shared_config` and add the following configuration to use either an ENUM server (recommended) or an ENUM file:

```
enum_server=<IP addresses of enum servers>
enum_file=<location of enum file>
```

If you use the ENUM file, enter the ENUM rules in the JSON format (shown above). If you are using the enhanced node management framework provided by `clearwater-etcd`, and you use `/etc/clearwater/enum.json` as your ENUM filename, you can automatically synchronize changes across your deployment by running `sudo cw-upload_enum_json` after creating or updating the file. In this case, other Sprout nodes will automatically download and use the uploaded ENUM rules.

It's possible to configure Sprout with secondary and tertiary ENUM servers, by providing a comma-separated list (e.g. `enum_server=1.2.3.4,10.0.0.1,192.168.1.1`). If this is done:

- Sprout will always query the first server in the list first
- If this returns SERVFAIL or times out (which happens after a randomised 500ms-1000ms period), Sprout will resend the query to the second server
- If this returns SERVFAIL or times out, Sprout will resend the query to the third server
- If all servers return SERVFAIL or time out, the ENUM query will fail

Each component of Sprout (I-CSCF, S-CSCF and BGCF) will perform ENUM lookups independently and at different points in call processing:

- The I-CSCF will perform an ENUM lookup during terminating processing if the Request URI represents a phone number.
  - If the result of this ENUM lookup is also a URI that represents a phone number, does not contain number portability information and is not in the HSS, another ENUM lookup is performed on the new URI.
- The S-CSCF will perform an ENUM lookup at the end of originating processing if the Request URI represents a phone number, regardless of whether there is number portability information in the URI.
- The BGCF will perform an ENUM lookup on receiving a request if the Request URI represents a phone number, regardless of whether there is number portability information in the URI.

The presence of number portability data and the `npdi` parameter do not affect whether an ENUM lookup is performed, but if number portability data is present then it is used for routing.

Voxbone

Voxbone provides local, geographical, mobile and toll free phone numbers and converts incoming calls and SMSs to SIP INVITE and MESSAGE flows.

Clearwater supports these flows, and this document describes how to configure Voxbone and Clearwater to work together, and then how to test and troubleshoot.

Configuration

There are 3 configuration steps.

1. Configure Clearwater's IBCF to trust Voxbone's servers.
2. Create a subscriber on Clearwater.
3. Associate the subscriber with a number on Voxbone.

Configuring IBCF

An IBCF (Interconnection Border Control Function) interfaces between the core network and one or must trusted SIP trunks. We will configure Voxbone to send all its traffic to the IBCF, but we first need to configure the IBCF to accept traffic from Voxbone.

Clearwater's bono component acts as an IBCF. To make things simpler later on, you should configure *all* of your bono nodes to act as IBCFs.

The IP addresses you need to specify as trusted peers are

- all the Voxbone signaling IP addresses - as of writing, these are
  - 81.201.82.45
  - 81.201.84.195
  - 81.201.83.45
  - 81.201.86.45
  - 81.201.85.45
- the Voxbone SMS IP addresses - as of writing, these are
  - 81.201.82.10
  - 81.201.82.11
  - 81.201.82.12.

The process for configuring trusted peers is described as part of the IBCF documentation.

If you don't want to configure all your bono nodes as IBCF nodes, you can do so, but will need to take extra steps when configuring Voxbone to ensure that calls are routed to the correct node.

Creating a subscriber

Creating a subscriber for Voxbone use is done exactly as you would normally do so.

Associating the subscriber on Voxbone

For each telephone number you own, Voxbone allows you to configure a SIP URI to which incoming calls will be sent.

- If, as recommended earlier, all your bono nodes are configured as IBCFs, you can simply use the SIP URI of the subscriber you created above.
- If only one (or a subset) of your bono nodes are configured as IBCFs, you must ensure that calls are routed to this node by replacing the domain name of the SIP URI with the IP address or domain name of the IBCF node. For example, if your SIP URI was `1234567890@example.com` but your IBCF node was `1.2.3.4`, you must use `1234567890@1.2.3.4`.

If you are also using the VoxSMS service for receiving SMSes, you additionally need to configure a VoxSMS SIP URI using the same SIP URI as above.

Testing

The Voxbone web UI allows you to make a test call. Alternatively, you can make a normal call through the PSTN to your Voxbone number.

The only way to test SMS function is to actually send an SMS to your Voxbone number.

Troubleshooting

If your call or SMS doesn't get through, there are three things to check.

- Does the SIP message actually reach the IBCF? You can check this using network capture (e.g. `tcpdump`) on the IBCF? If not, the problem is likely to either be that the SIP URI's domain doesn't resolve to your IBCF, or that security groups or firewall rules are blocking traffic to it.
- Is the SIP message trusted? You can again check this with network capture - if you see a `403 Forbidden` response, this indicates that the IBCF does not trust the sender of the message. Check the `trusted_peers` entry in your `/etc/clearwater/user_settings` file and that you have restarted bono since updating it.
- Would the call go through if it were sent on-net? Try making a call to the subscriber from another subscriber on Clearwater and check that it goes through (or follow the standard troubleshooting process).

SNMP Statistics

Clearwater provides a set of statistics about the performance of each Clearwater nodes over SNMP. Currently, this is available on Bono, Sprout, Vellum and Dime nodes, and the MMTel, Call Diversion, Memento and Gemini Application Server nodes. A number of the statistics we offer detail the current state of the system in use, and as such are termed 'Stateful' statistics. These statistics will be marked as '(stateful)' below, and more information on them can be found here

Configuration

These SNMP statistics require:

- the clearwater-snmpd package to be installed for all node types
- the clearwater-snmp-handler-astaire package to be installed for Vellum nodes

These packages will be automatically installed when installing through the Chef automation system; for a manual install, you will need to install the packages with `sudo apt-get install`.

Usage

Clearwater nodes provide SNMP statistics over port 161 using SNMP v2c and community `clearwater`. The MIB definition file is included in the clearwater-snmp-alarm-agent package, copied to `/usr/share/clearwater/mibs/PROJECT-CLEARWATER-MIB` on installation. Alternatively, the MIB can be built from a checkout of the clearwater-snmp-handlers repository - see the repo readme for details.

Our SNMP statistics are provided through plugins or subagents to the standard SNMPd packaged with Ubuntu, so querying port 161 (the standard SNMP port) on a Clearwater node will provide system-level stats like CPU% as well as any available Clearwater stats.

To load the MIB file, allowing you to refer to MIB objects by name, first place it in the `~/.snmp/mibs` directory. To load the MIB file for just the current session, run `export MIBS=+PROJECT-CLEARWATER-MIB`. To load the MIB file every time, add the line `mibs +PROJECT-CLEARWATER-MIB` to a snmp.conf file in the `~/.snmp` directory.

If a statistic is indexed by time period, then it displays the relevant statistics over:

- the previous five-second period
- the current five-minute period

- the previous five-minute period

For example, a stat queried at 12:01:33 would display the stats covering:

- 12:01:25 - 12:01:30 (the previous five-second period)
- 12:00:00 - 12:01:33 (the current five-minute period)
- 11:55:00 - 12:00:00 (the previous five-minute period)

All latency values are in microseconds.

Many of the statistics listed below are stored in SNMP tables (although the MIB file should be examined to determine exactly which ones). The full table can be retrieved by using the `snmptable` command. For example, the Initial Registrations table for Sprout can be retrieved by running: `snmptable -v2c -c clearwater <ip> PROJECT-CLEARWATER-MIB::sproutInitialRegistrationTable`

The individual table elements can be accessed using: `snmpget -v2c -c clearwater <ip> <table OID>.1.<column>.<row>`

For example, the Initial Registration Stats table has an OID of `.1.2.826.0.1.1578918.9.3.9`, so the number of initial registration attempts in the current five-minute period can be retrieved by: `snmpget -v2c -c clearwater <ip> .1.2.826.0.1.1578918.9.3.9.1.2.2` or by: `snmpget -v2c -c clearwater <ip> PROJECT-CLEARWATER-MIB::sproutInitialRegistrationAttempts.scopeCurrent5MinutePeriod`

The `snmpwalk` command can be used to discover the list of queryable OIDs beneath a certain point in the MIB tree. For example, you can retrieve all of the entries in the Sprout Initial Registrations table using: `snmpwalk -v2c -c clearwater <ip> PROJECT-CLEARWATER-MIB::sproutInitialRegistrationTable`

Running `snmpwalk -v2c -c clearwater <ip> PROJECT-CLEARWATER-MIB::projectClearwater` will output a very long list of all available Clearwater stats.

Note that running an 'snmpget' on a table OID will result in a "No Such Object available on this agent at this OID" message.

Bono statistics

Bono nodes provide the following statistics:

- The standard SNMP CPU and memory usage statistics (see http://net-snmp.sourceforge.net/docs/mibs/ucdavis.html for details)
- The average latency, variance, highest latency and lowest latency for SIP requests, indexed by time period.
- The number of parallel TCP connections to each Sprout node.
- The number of incoming requests, indexed by time period.
- The number of requests rejected due to overload, indexed by time period.
- The average request queue size, variance, highest queue size and lowest queue size, indexed by time period.

Sprout statistics

Sprout nodes provide the following statistics:

- The standard SNMP CPU and memory usage statistics (see http://net-snmp.sourceforge.net/docs/mibs/ucdavis.html for details)
- The average latency, variance, highest latency and lowest latency for SIP requests, indexed by time period.

- The average latency, variance, highest latency and lowest latency for requests to Homestead, indexed by time period.
- The average latency, variance, highest latency and lowest latency for requests to Homestead's `/impi/<private ID>/av` endpoint, indexed by time period.
- The average latency, variance, highest latency and lowest latency for requests to Homestead's `/impi/<private ID>/registration-status` endpoint, indexed by time period.
- The average latency, variance, highest latency and lowest latency for requests to Homestead's `/impu/<public ID>/reg-data` endpoint, indexed by time period.
- The average latency, variance, highest latency and lowest latency for requests to Homestead's `/impu/<public ID>/location` endpoint, indexed by time period.
- The average latency, variance, highest latency and lowest latency for requests to Homer, indexed by time period.
- The number of attempts, successes and failures for AKA authentications on register requests, indexed by time period (AKA authentication attempts with a correct response but that fail due to the sequence number in the nonce being out of sync are counted as successes). Also (for convenience) the percentage of such authentications that were successful.
- The number of attempts, successes and failures for SIP digest authentications on register requests, indexed by time period (authentication attempts with a correct response but that fail due to being stale are counted as failures). Also (for convenience) the percentage of such authentications that were successful.
- The number of attempts, successes and failures for authentications on non-register requests, indexed by time period. Also (for convenience) the percentage of such authentications that were successful.
- The number of attempts, successes and failures for initial registrations, indexed by time period (registrations that fail due to failed authentication are counted in the authentication stats and not here). Also (for convenience) the percentage of such registrations that were successful.
- The number of attempts, successes and failures for re-registrations, indexed by time period (registrations that fail due to failed authentication are counted in the authentication stats and not here). Also (for convenience) the percentage of such re-registrations that were successful.
- The number of attempts, successes and failures for de-registrations, indexed by time period (registrations that fail due to failed authentication are counted in the authentication stats and not here). Also (for convenience) the percentage of such de-registrations that were successful.
- The number of attempts, successes and failures for third-party initial registrations, indexed by time period (registrations that fail due to failed authentication are counted in the authentication stats and not here). Also (for convenience) the percentage of such registrations that were successful.
- The number of attempts, successes and failures for third-party re-registrations, indexed by time period (registrations that fail due to failed authentication are counted in the authetication stats and not here). Also (for convenience) the percentage of such re-registrations that were successful.
- The number of attempts, successes and failures for third-party de-registrations, indexed by time period (registrations that fail due to failed authentication are counted in the authentication stats and not here). Also (for convenience) the percentage of such de-registrations that were successful.
- The number of requests routed by the S-CSCF according to a route pre-loaded by an app server, indexed by time period.
- The number of parallel TCP connections to each Homestead service.
- The number of parallel TCP connections to each Homer node.
- The number of incoming SIP requests, indexed by time period.
- The number of requests rejected due to overload, indexed by time period.

- The average request queue size, variance, highest queue size and lowest queue size, indexed by time period.
- The number of attempts, successes and failures for incoming SIP transactions for the ICSCF, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for outgoing SIP transactions for the ICSCF, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures to establish terminating sessions at the I-CSCF, indexed by time period. Also (for convenience) the percentage of such attempts that were successful. Each INVITE received by the I-CSCF (from the originating S-CSCF) is counted as a terminating session attempt. Such an attempt is considered successful if the I-CSCF responds with a 180 RINGING or 200 OK.
- The number of attempts, successes and failures to establish terminating sessions at the I-CSCF with success measured from the perspective of the network, indexed by time period. Also (for convenience) the percentage of such attempts that were successful. This is the same as the previous set of statistics, but now sessions are considered to be established successfully if either: the I-CSCF responds with a 180 RINGING or 200 OK; the session is canceled by the originating party before being established; the session is rejected with 486 BUSY HERE, 600 BUSY EVERYWHERE, 404 NOT FOUND or 484 ADDRESS INCOMPLETE.
- The number of attempts, successes and failures for incoming SIP transactions for the SCSCF, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for outgoing SIP transactions for the SCSCF, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for incoming SIP transactions for the BGCF, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for outgoing SIP transactions for the BGCF, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The permitted request rate (PRR) is an estimate for the sustainable request rate without causing large latency. Sprout provides a weighted average permitted request rate, variance, highest PRR, and lowest PRR, indexed by time period.
- The value of the smoothed latency at the last permitted request rate update.
- The value of the target (maximum permissible) latency at the last permitted request rate update.
- The number of penalties experienced at the last permitted request rate update.
- The current permitted request rate.
- The number of incoming INVITE transactions for the S-CSCF that were cancelled before a 1xx response was seen, indexed by time period.
- The number of incoming INVITE transactions for the S-CSCF that were cancelled after a 1xx response was seen, indexed by time period (these INVITE cancellation statistics can be used to distinguish between the case where an INVITE was cancelled because the call rang but wasn't answered and the case where it failed due to network issues and never got through in the first place).
- The number of additional INVITEs sent due to public identities having multiple registered bindings.
- The count, average, variance, and high and low watermarks for originating audio session setup time at the S-CSCF, indexed by time period. For the purposes of these stats a call is considered to be an audio call if video is not specified in the SDP on the initial INVITE. The session setup time is measured as the time between receiving the originating INVITE and sending the first successful response (e.g. 180 RINGING or 200 OK).
- The count, average, variance, and high and low watermarks for originating video session setup time at the S-CSCF, indexed by time period. For the purposes of these stats a call is considered to be a video call if video is specified in the SDP on the initial INVITE. The session setup time is measured as the time between receiving the originating INVITE and sending the first successful response (e.g. 180 RINGING or 200 OK).

- The number of Shared iFC set IDs retrieved from the HSS where the S-CSCF has no matching set of iFCs, indexed by time period.
- The number of initial requests that had no matching iFCs to apply, indexed by time period.
- The number of initial requests that attempted to use fallback iFCs but had no matching ones, indexed by time period.
- The number of requests rejected because an identity was barred.

Vellum statistics

Vellum nodes provide the following statistics:

- The standard SNMP CPU and memory usage statistics (see http://net-snmp.sourceforge.net/docs/mibs/ucdavis.html for details).
- The number of Memcached buckets needing to be synchronized and buckets already resynchronized during the current Astaire resynchronization operation (overall, and for each peer).
- The number of Memcached entries, and amount of data (in bytes) already resynchronized during the current Astaire resynchronization operation.
- The transfer rate (in bytes/second) of data during this resynchronization, over the last 5 seconds (overall, and per bucket).
- The number of remaining nodes to query during the current Chronos scaling operation.
- The number of timers, and number of invalid timers, processed over the last 5 seconds.
- The total number of timers being managed by a Chronos node at the current time.
- The weighted average of total timer count, variance, highest timer count, lowest timer count, indexed by time period.
- The average count, variance, and high and low watermarks for the number of registrations, indexed by time period. (stateful)
- The average count, variance, and high and low watermarks for the number of bindings, indexed by time period. (stateful)
- The average count, variance, and high and low watermarks for the number of subscriptions, indexed by time period. (stateful)
- The number of registrations active at the time queried. (stateful)
- The number of bindings active at the time queried. (stateful)
- The number of subscriptions active at the time queried. (stateful)
- The average count, variance, and high and low watermarks for the number of calls, indexed by time period. (stateful)
- The number of calls active at the time queried. (stateful)

Dime Statistics

Dime nodes provide the following statistics:

- The standard SNMP CPU and memory usage statistics (see http://net-snmp.sourceforge.net/docs/mibs/ucdavis.html for details)
- The average latency, variance, highest call latency and lowest latency on HTTP requests, indexed by time period.

- The average latency, variance, highest latency and lowest latency on the Cx interface, indexed by time period.
- The average latency, variance, highest latency and lowest latency on Multimedia-Auth Requests on the Cx interface, indexed by time period.
- The average latency, variance, highest latency and lowest latency on Server-Assignment, User-Authorization and Location-Information Requests on the Cx interface, indexed by time period.
- The number of incoming requests, indexed by time period.
- The number of requests rejected due to overload, indexed by time period.
- The total number of Diameter requests with an invalid Destination-Realm or invalid Destination-Host, indexed by time period.
- The number of Multimedia-Authorization-Answers with a given result-code received over the Cx interface, indexed by time period.
- The number of Server-Assignment-Answers with a given result-code received over the Cx interface, indexed by time period.
- The number of User-Authorization-Answers with a given result-code received over the Cx interface, indexed by time period.
- The number of Location-Information-Answers with a given result-code received over the Cx interface, indexed by time period.
- The number of Push-Profile-Answers with a given result-code sent over the Cx interface, indexed by time period.
- The number of Registration-Termination-Answers with a given result-code sent over the Cx interface, indexed by time period.

Call Diversion App Server Statistics

Call Diversion App Server nodes provide the following statistics:

- The number of attempts, successes and failures for incoming SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for outgoing SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.

Memento App Server Statistics

Memento App Server nodes provide the following statistics:

- The number of attempts, successes and failures for incoming SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for outgoing SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.

MMTel App Server Statistics

MMTel App Server nodes provide the following statistics:

- The number of attempts, successes and failures for incoming SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.

- The number of attempts, successes and failures for outgoing SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.

Gemini App Server Statistics

Gemini App Server nodes provide the following statistics:

- The number of attempts, successes and failures for incoming SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.
- The number of attempts, successes and failures for outgoing SIP transactions, indexed by time period and request type. Also (for convenience) the percentage of such transactions that were successful.

Stateful Statistics

Clearwater is designed to be as stateless as possible, simplifying redundancy, and making it ideal for virtualised and cloud-based deployment. However, through use of our distributed timer store, Chronos, Clearwater is able to maintain a number of stateful statistics. As components use timers to maintain stateful functionality without actually remaining independently aware of state, we can form statistics based on these, rather than needing each main process to maintain and report state. Currently Vellum nodes expose stateful statistics. To see the full list of statistics available, check here; stateful statistics are marked as '(stateful)' to distinguish them.

Configuration

These statistics are accessible in the same manner as our standard SNMP statistics. Detail on how to configure nodes and access the statistics can be found here.

Usage

As these statistics are representative of deployment wide state, their usage is slightly different to the standard per-node SNMP statistics. They key difference is that, due to the redundancy design in Chronos, the statistics are subject to a replication factor (which is by default set to 2). To accurately report the statistics from a whole deployment, the individual values should be gathered from each node in the deployment, and then the sum total should be divided by the replication factor.

e.g. Consider a deployment with three Vellum nodes. If ten registrations are set up, the nodes might report the following statistics:

- Node 1 :- 6 Registrations
- Node 2 :- 7 Registrations
- Node 3 :- 7 Registrations

This does not mean that the number of registrations has doubled, nor is it representative of the number of registrations that each node actually handled. It is simply directly the number of timers held on each node tagged as registrations. To then calculate the number of registrations actually active in the deployment, one takes the total and divides it by the replication factor, in this example 2.

(Node 1 + Node 2 + Node 3) / Replication-factor = (6 + 7 + 7) / 2 = 10 Active registrations

Note: These statistics may be inaccurate in systems with a failing node. As such they should be considered unreliable while any alarms are raised, particularly those relating to Chronos.

Technical details

For a more technical overview of how these statistics are handled within Chronos, see here.

SNMP Alarms

Clearwater nodes report errors over the SNMP interface when they are in an abnormal state, and clear them when they return to normal. They only report errors relating to that node - not errors relating to the whole deployment, which is the role of an external monitoring service.

Error indications come in two forms:

- For clearly-defined errors not based on thresholds, the Clearwater node sends an SNMP notification to a configured external SNMP manager. This notification can be in one of two fomats, enterprise-specific or RFC 3877 - see below for details.
- For errors based on a threshold set on a statistic (such as latency targets or number of failed connections), the Clearwater node exposes that statistic over SNMP. A downstream statistics aggregator from the Management and Orchestration (MANO) layer monitors these statistics, compares them to its configured thresholds, and raises alarms on that basis.

To configure this feature, set `snmp_ip` and `snmp_notification_types` as described in the configuration guide.

Alarm Formats

Clearwater Alarm MIB

The simplest way to consume Clearwater alarms is to use our enterprise alarm MIB. When this is used, Clearwater nodes will send traps with the following fields:

- **CLEARWATER-ENTERPRISE-MIB::clearwaterInfoVersion** - the version of the MIB (e.g. '201608081100')
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapDisplayName** - the summary name of the alarm (e.g. QUEUE_MANAGER_PROCESS_FAIL). This is the same for all severities (e.g. CLEARED and CRITICAL severities have the same name)
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapAlarmOID** - the OID of the alarm, for correlation purposes (e.g. ALARM-MIB::alarmModelNotificationId.0.9000.2)
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapSeverity** - the severity of this alarm (e.g. CLEARED)
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapDescription** - a short description of the alarm (e.g. 'Queue manager: Process failure cleared')
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapDetails** - a longer description of the alarm (e.g. 'The queue manager process has been restored to normal operation.')
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapCause** - the likely cause of this notification (e.g. 'The queue manager process has been restored to normal operation. The previously issued alarm has been cleared.')
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapEffect** - the effect on the system (e.g. 'Changes to shared config will be synchronized across the cluster.')
- **CLEARWATER-ENTERPRISE-MIB::alarmTrapAction** - the suggested next action for the operator (e.g. 'No action')

- **CLEARWATER-ENTERPRISE-MIB::alarmTrapHostname** - the hostname of the alarmed node (e.g. 'sprout-1.example.com')

RFC 3877

If your EMS already supports alarms in RFC 3877 format, it may make sense to use these instead of our enterprise alarms. The RFC 3877 model is that the SNMP INFORM messages are minimal, just containing the OID of a row in an SNMP table - the EMS can then learn the details of the alarm by querying that SNMP table.

The EMS must support the following capabilities of RFC 3877 to obtain the brief description, detailed description, and severity for the alarm:

- The EMS must be configured to catch the SNMP INFORM messages used to report alarms from Clearwater. It is also recommended that the EMS must display the alarm information provided by the following MIBs.
- Upon receiving a SNMP INFORM message from Clearwater the EMS can obtain the alarm data by the following:
  - The EMS must retrieve the AlarmModelEntry MIB table data associated with the current SNMP INFORM message from Clearwater. This MIB provides the active/clear state, alarm description, and a table index for the ituAlarmEntry MIB.
  - The EMS must the retrieve the ituAlarmEntry MIB table data associated with the current alarm from Clearwater. This MIB provides the alarm severity and the additional detailed alarm description.

RFC 3877 alarms have a couple of limitations compared to our enterprise alarms:

- RFC 3877 doesn't define fields for cause, effect or action, so these are not accessible - only the details and description are present.
- RFC 3877 defines a 255-character limit for details and description, so these fields may be briefer than in the enterprise MIB (which allows 4,096 characters).

Alarm Models

The alarm models used by Clearwater are defined in alarmdefinition.h.

Alarm Resiliency

Since prompt, accurate notifications of alarms are important for running a reliable telecoms network, Clearwater is designed so that if an EMS or an internal Clearwater component responsible for alarm notifications fails and recovers, the EMS will learn about any alarms raised by Clearwater during the outage, rather than those alarms just being lost. The exact behaviour is as follows:

- On startup (including after reboots), Clearwater nodes will detect whether each of their alarms should be raised or not (and at what severity), and send corresponding SNMP INFORMs to the EMS. This will happen within a minute.
  - If there is no call traffic, the correct state of some alarms cannot be determined – in this case, the node will neither raise nor clear an alarm until there is call traffic and it can determine the correct state.
- If the EMS suffers an outage, but remembers previous alarm state when it recovers, no operator intervention is required. Clearwater nodes send SNMP INFORMs, which require acknowledgement by the EMS, and these will be retransmitted until they are acknowledged.
  - Note that if the alarm changes state multiple times during an outage (e.g. if it is raised but then cleared, or raised once at major severity and then again at critical severity), only the latest state will be transmitted to the EMS on connection recovery.

- If the EMS suffers an outage and loses previous alarm state, there are two ways to recover:
  - The process responsible for sending SNMP notifications, the Alarm Agent, keeps track of all the currently active error states in a table called the Alarm Active Table (defined in RFC 3877). Upon restart, an EMS can read this table and learn about any SNMP INFORMs it may have missed in its downtime.
  - If an EMS does not support reading the Active Alarm Table, the operator can still recover the SNMP INFORMs by running `/usr/share/clearwater/bin/sync_alarms.py` on each node. This will cause SNMP notifications to be resent to the EMS.

Clearwater nodes also regularly re-detect and re-transmit alarm state internally. This is so that if the alarm agent is failed when an alarmable condition is detected, the alarm will still reach the EMS less than a minute after the component recovers. If the alarm agent fails, it will be automatically restarted, but it will lose its alarm state. However, each internal component which detects alarms will retransmit them on a 30-second timer – so within a minute, they will have retransmitted their alarm state to the alarm agent, it will have sent corresponding SNMP INFORMs to the EMS, and both the alarm agent and the EMS will have a complete picture of the current alarm state.

Radius Authentication

It is possible to authenticate users for SSH/SFTP access by using the RADIUS authentication protocol to contact a third party RADIUS authentication server. The following will explain how to install and configure this capability on a Clearwater node, and how to go about using it.

The authentication process

On attempting to access a node via SSH or SFTP, a user is either expected to use a key, or provide a password to verify their identity. This process requires a locally provisioned set of authentication details for each user that the sshd process compares to the provided credentials for verification. For password based authentication, enabling RADIUS means that all user accounts can be configured centrally on a RADIUS server (or in a database the RADIUS server can access). Each node can then pass on the user credentials provided to this server to complete the authentication process.

Sshd requires that the user attempting to access a node must exist locally. To allow the authentication process to complete correctly, any locally unknown user is mapped to a single configurable user (usually the system default user). As such, all RADIUS authenticated users will be acting as this user on the local node. For auditing purposes however, the username provided at login is recorded (e.g. in `/var/log/auth.log`, and the output of commands like `who`).

Prerequisites

The following conditions are assumed by this process:

- Your nodes are running on Ubuntu 14.04.
- Your nodes have access to both Clearwater and Ubuntu repositories.
- Your SSH configuration allows password authentication and PAM (the correct configuration will be detailed below).
- You have access to a third-party RADIUS server (such as freeRADIUS).
- Your firewall allows UDP traffic to the above server on port 1812.

Installation

Package installation

Install the Clearwater RADIUS authentication package:

```
sudo apt-get install clearwater-radius-auth
```

Configuration

libnss-ato configuration

You need to create configuration for the all-to-one module, libnss-ato, which is used to map your RADIUS authenticated users onto a locally provisioned user. A template of this configuration is provided in `/etc/libnss-ato.conf.TEMPLATE`. You will need to create the file `/etc/libnss-ato.conf`, and provide an entry in the same format as in the template file. For most users, the template can be copied directly, simply running `sudo cp /etc/libnss-ato.conf.TEMPLATE /etc/libnss-ato.conf`; this will map locally unknown users to the default linux user (UID 1000). If you wish to configure the module to use a different user, the entry should match how the user appears in `/etc/passwd`. Only a single user mapping is configurable, and so only the first line of the configuration file is parsed.

RADIUS server configuration

The details of your RADIUS server will need to be entered into `/etc/pam_radius_auth.conf`. This file provides an example of how entries should be structured: * Multiple entries are allowed, but each must be on a new line. * Each line consists of three fields:

```
server[:port] (The default is port 1812. All traffic will be UDP)
secret
[timeout] (Default timeout is 3 seconds)
```

- The secret is shared between each client and the server to allow simple encryption of passwords. The secret must match the entry for the client in the RADIUS server configuration.
- Both the port and timeout entries are optional. We recommend a relatively small timeout value (e.g. 3 seconds), as in the case that your RADIUS server becomes uncontactable users will have to wait the full duration of all configured timeouts before falling back to local password based authentication. Authentication by SSH-key does not enter this authentication path, and so timeout values will not impact SSH-key based access.

Sshd configuration

Your sshd configuration must allow password authentication, and use of PAM. If you are unsure, check that the `PasswordAuthentication` and `UsePAM` entries in `/etc/ssh/sshd_config` are set to `yes`. Any changes to ssh configuration will require the ssh process to be restarted before coming into effect.

You must ensure that your firewall/security groups allow UDP traffic to the RADIUS server on port 1812.

Usage

Once the above is installed and configured, you will need to enable the RADIUS authentication process. To do this, simply run `sudo /usr/share/clearwater-radius-auth/bin/`

`enable-radius-authentication`. Once enabled, any user provisioned in the RADIUS server can attempt SSH or SFTP access to the configured node, and on providing their password they will be authenticated against the details held on the RADIUS server, and logged in, acting as the node default user. Commands such as `who` or `last` will output the username supplied at login, and this will also be recorded in the auth log `/var/log/auth.log`.

Any users provisioned locally on the node will see no change to their authentication experience. By default, RADIUS authentication is set to be a sufficient, but not required condition. As such, failing to authenticate against the server credentials will cause the authentication attempt to fall back to checking locally provisioned details. See below for further details on configuration options.

Troubleshooting

- If you are not seeing any traffic reaching your RADIUS server, and entries in `/var/log/auth.log` state that no RADIUS server was reachable, re-check the RADIUS server entry in `/etc/pam_radius_auth.conf`, and ensure that your firewall is configured to allow UDP traffic to the RADIUS server on port 1812.
- If your RADIUS server is rejecting authentication requests, ensure that the server is configured correctly.

Removal

If you simply want to disable RADIUS authentication on a node, run `sudo /usr/share/clearwater-radius-auth/bin/disable-radius-authentication`.

To properly remove clearwater-radius-auth, and the components it brings with it, run the following:

```
sudo apt-get purge clearwater-radius-auth
sudo apt-get purge libpam-radius-auth
sudo apt-get purge libnss-ato
sudo rm -f /etc/libnss-ato.conf
```

This will remove all configuration put in place by the installation. Should your configuration become corrupt, purging and re-installing the associated module will re-instate the correct configuration.

Further configuration

This section details the configuration put in place by the installation. It is highly recommended that these be left as their defaults. The following is for information purposes only.

libnss-ato.conf

The libnss-ato configuration file is found at `/etc/libnss-ato.conf`. Users will need to manually create the file on their first installation. The file contains an entry specifying the user identity to which unknown users are mapped. A template of the configuration is provided at `/etc/libnss-ato.conf.TEMPLATE`. It will look like the following:

```
radius_authenticated_user:x:1000:1000:radius_authenticated_user:/tmp:/bin/bash
```

For most installations, copying the template across to create the configuration file will be sufficient. This will map unknown users to the default user, UID 1000.

Only the first line of this file is parsed. The user entry is the same format as is found in `/etc/passwd`. Replacing this file with a different user entry will map unknown users to the entry provided.

pam.d/sshd and pam.d/login

The PAM configuration file for the sshd and login processes are found at `/etc/pam.d/sshd` and `/etc/pam.d/login` respectively. As part of the installation, the 3 lines around `auth sufficient pam_radius_auth.so` are added at the top of these files, configuring PAM to attempt RADIUS authentication before other methods. It will look like the following:

```
# PAM configuration for the Secure Shell service
# +clearwater-radius-auth
auth sufficient pam_radius_auth.so
# -clearwater-radius-auth
# Standard Un*x authentication.
```

It is strongly recommended that users do not modify these entries. Further information on this configuration can be found at FreeRADIUS.

nsswitch.conf

The NSS configuration file is found at `/etc/nsswitch.conf`. After installation, the top three entries in this file will look as follows:

```
passwd:   compat ato
group:    compat
shadow:   compat ato
```

Modifications to the NSS configuration make it check the libnss-ato component for a user mapping if no local user is found. The addition of `ato` at the end of both the `passwd` and `shadow` entries provides this function. Removal of these addition will disable the user mapping, and break RADIUS authentication.

Application Server Guide

You can add new call features or functionality to calls by adding an application server. Clearwater supports application servers through the standard IMS interface ISC. This article explains the features and limitations of this support. See Configuring an Application Server for details of how to configure Clearwater to use this function.

What is an application server?

An application server (AS) is a server which is brought into or notified of a call by the network in order to provide call features or other behaviour.

In IMS, an application server is a SIP entity which is invoked by the S-CSCF for each dialog, both originating and terminating, as configured by the initial filter criteria (iFCs) of the relevant subscribers. The application server may reject or accept the call itself, redirect it, pass it back to the S-CSCF as a proxy, originate new calls, or perform more complex actions (such as forking) as a B2BUA. It may choose to remain in the signaling path, or drop out.

The application server communicates with the IMS core via the ISC interface. This is a SIP interface. The details are given in 3GPP TS 24.229, especially s5.4 (the S-CSCF side) and s5.7 (the AS side). Further useful information can be found in 3GPP TS 23.228. Section references below are to 3GPP TS 24.229 unless otherwise specified.

For a more detailed look at the ISC interface, and how it is implemented in Clearwater, see the ISC interface guide in the Sprout developer documentation.

Clearwater interfaces

In Clearwater most S-CSCF function, including the ISC interface, is implemented in Sprout. Sprout invokes application servers over the ISC interface, as specified in the iFCs.

- Per the IMS specs, this invocation occurs on dialog-initiating requests such as INVITE, SUBSCRIBE, MESSAGE, etc, and according to the triggers within the iFCs (s5.4.3.2, s5.4.3.3):
  - When specified, Sprout will route the message to the AS; the AS can either route it onward, act as a B2BUA for e.g. call diversion, or give a final response.
  - Clearwater has full support for chained iFCs. The original dialog is tracked by Sprout using an ODI token in the Route header. This support includes support for forking ASs at any point in the chain.
  - Service trigger points (i.e., conditions) are implemented, including SIP method, SIP headers, SDP lines, registration parameters, and request URIs.
  - Service trigger points can have session case configuration, allowing the AS to only be invoked on originating calls or on terminating calls.
- No per-AS configuration is required; ASs are invoked simply by their URI appearing in the iFCs.
- AS invocation also occurs on REGISTER - this is called third-party registration (3GPP TS 24.229 s5.4.1.7 and 7A):
  - When a UE registers with Sprout, if the iFCs require it, it passes a third-party registration onto an AS.
  - Message body handling for third-party registration, per 3GPP TS 24.229 s5.4.1.7A: including optionally service info, a copy of the registration, and a copy of the response.
  - Network-initiated deregister. If the third-party registration fails and the iFC requests it, we must deregister the UE.

Supported SIP headers

The following SIP headers are supported on the ISC interface:

- All standard RFC3261 headers and relevant flows, including `To`, `From`, `Route`, `Contact`, etc. Also `Privacy` (per RFC3323) and `Path` (per RFC3327).
- `P-Asserted-Identity` - bare minimal support only: we set it to the same value as `From:` on the outbound ISC interface, and never strip it. Full support will be added in future phases. This header is needed by originating ASs, particularly in cases where privacy is requested (see RFC3325). Full support would involve setting it in P-CSCF (bono), and ensuring it is set/stripped in the right places. The proposed minimal support has the limitation that ASs which don't understand `P-Served-User` won't work correctly when privacy is invoked. See 3GPP TS 24.229 s5.7.1.3A.
- `P-Served-User`. This is needed for proper support of AS chains, to avoid service-interaction tangles. It is set by Sprout and set/stripped in the appropriate places.

Points of Note

- Trust:
  - Some ISC signaling is trust-dependent. For Clearwater, all ASs are trusted - we think support for untrusted ASs is unlikely to be required.
- IP Connectivity
  - We assume that there is no NAT (static or otherwise) between the AS and the Clearwater core.

Current Spec Deltas

As of June 2015, Clearwater has the following limitations on the ISC interface:

- Change of request URI:
  - 3GPP TS 24.229 s5.4.3.3 step 3 allows a terminating AS to change the request URI to another URI that matches it (i.e., is canonically equal or an alias) without interrupting the interpretation of the iFCs.
  - Clearwater only supports this for URIs which are canonically equal; it does not support changing to an alias URI (i.e., a different public identity belonging to the same alias group of the same private identity, per 3GPP TS 24.229 s3.1, 3GPP TS 29.228 sB.2.1).
- `Request-Disposition: no-fork` (3GPP TS 24.229 s5.4.3.3 step 10, s5.7.3, RFC 3841)
  - Clearwater ignores this directive - it always INVITEs all registered endpoints.

Future phases

The following features may be implemented by Clearwater in future phases:

- Billing: both billing headers within ISC, and AS communicating with the billing system via Rf/Ro.
- `P-Access-Network-Info`: should be passed (from the UE) and stripped in the appropriate places, and possibly set by bono.
- `History-Info` draft-ietf-sipcore-rfc4244bis (not RFC4244, which is inaccurate wrt real implementations).
- `user=dialstring` handling (RFC4967). The specs (3GPP TS 24.229 s5.4.3.2, esp step 10) are quite clear that this is handed off to an AS or handled in a deployment-specific way, as for various other URI formats, so there is nothing to do here.
- `P-Asserted-Service` / `P-Preferred-Service` (RFC6050, TS 23.228 s4.13), i.e., IMS Communication Services and ICSIs.
- IMS debug package, IMS logging.
- Support for untrusted ASs.
- Support for terminating ASs changing to an alias URI.
- Support for `Request-Disposition: no-fork`, which should pick a single endpoint to INVITE.

Exclusions

Clearwater implements the ISC interface only. An AS may also expect to communicate over several other interfaces:

- AS interrogates HSS over Sh. If the AS requires this it should access the HSS directly, bypassing Clearwater's Homestead cache. It is not supported in deployments without an HSS.
- AS interrogates SRF over Dh. Typical Clearwater deployments do not include an SRF.
- UE retrieves and edits AS configuration via Ut. An AS is free to provide this or any other configuration interface it chooses. Homer does not provide a generic Ut interface for ASs to store configuration information.

The built-in MMTEL application server

Clearwater has a built-in application server, `mmtel.<deployment-domain>`, which implements a subset of the MMTEL services defined in GSMA PRD IR.92, ETSI TS 129.364 and 3GPP TS 24.623:

- Originating Identification Presentation (OIP)

- Originating Identification Restriction (OIR)
- Communication Diversion (CDIV)
- Communication Barring (CB)

Note that Clearwater is also able to support a number of other IR.92 services, either through inherent support (by transparently passing messages between UEs) or through integration with external application servers. See the IR.92 Supplementary Services document for more information.

The built-in MMTEL application server is invoked only for calls configured to use it. To use it, simply configure a subscriber's iFCs to indicate the use of `mmtel.<deployment-domain>` as an application server. The MMTEL application server can be used on its own, or as one of a chain of application servers handling a call. The default iFCs configured by Ellis specify that the built-in MMTEL application server should be used for all originating and terminating calls.

WebRTC support in Clearwater

Clearwater supports WebRTC clients. This article explains how to do this, using a common WebRTC client as an example.

This has been tested using Firefox 46 and some previous versions. It does not work in Google Chrome.

Be aware that the sipML5 client can't be logged in to multiple numbers simultaneously in the same browser.

Instructions

In the instructions below, <domain> is your Clearwater domain.

- Configure a line in ellis.
- Check the camera is not in use: check the camera light is off. If it is on, close the app that is using it (e.g., Skype, or a phone client).
- Go to sipML5 live demo.
- By default sipML5 uses a SIP<->WebRTC gateway run by sipml5.org. Clearwater supports WebRTC directly. To tell sipML5 to speak WebRTC directly to Clearwater:
  - Click on the "expert mode" button to open the "expert mode" tab, and fill in the following field:
    - WebSocket Server URL: ws://<domain>:5062
  - Click Save.
  - Go back to the calls tab.
- Fill in the fields as follows:
  - Display name: <whatever you like, e.g., your name>
  - Private identity: <phone number>@<domain>
  - Public identity: sip:<phone number>@<domain>
  - Password: <password as displayed in ellis>
  - Realm: <domain>
- Now click Log In. This sometimes takes a few seconds. If it fails (e.g., due to firewall behaviour), click Log Out then Log In again.

Some firewalls or proxies may not support WebSockets (the technology underlying WebRTC). If this is the case, logging in will not work (probably it will time out). Clearwater uses port 5062 for WebSockets.

Once you have this set up on two tabs, or two browsers, you can make calls between them as follows:

- Entering the phone number
- Press the Call button.
- Accept the local request to use the camera/mic.
- Remote browser rings.
- Accept the remote request to use the camera/mic.
- Click Answer.
- Talk.
- Click Hang up on either end.

Video can be disabled on the 'expert mode' tab.

Note: Clearwater does not transcode media between WebRTC and non-WebRTC clients, so such calls are not possible. Transcoding will be supported in future.

Known sipML5 bugs

At the time of writing, the following sipML5 bugs are known:

- Calls where one or both ends do not have a webcam do not always complete correctly. It is best to ensure both ends have a webcam, even if audio-only calls are being made.
- Hang up doesn't work - you have to hang up on both ends.

Using STUN and TURN

Note: The above uses a default (non-Clearwater) STUN server, and no TURN server. Clearwater has its own STUN and TURN servers which can be used to support clients behind NATs and firewalls.

If one or both clients are behind a symmetric firewall, you must use TURN. Configure this as follows:

- On the 'expert mode' tab, fill in the following field with your SIP username and password.
  - ICE Servers: [{url:"turn:<phone number>%40<domain>@<domain>",credential:"<password>"}]

Testing TURN

To test TURN when one or both of the clients are running under Windows 7, find the IP of the remote client, and block direct communication between them as follows:

- Control Panel > Windows Firewall > Advanced Settings > Inbound Rules > New Rule > Custom > All Programs > Any Protocol > specify remote IP address > Block the connection > (give it a name).
- Do the same for outbound.

IR.92 Supplementary Services

GSMA IR.92 defines a set of Supplementary Services (section 2.3). This document describes Clearwater's support for each.

The services fall into one of four categories.

- Supported by Clearwater's built-in MMTel application server
- Supported by Clearwater inherently (i.e. whether or not the MMTel application server is enabled) - these services rely on messages or headers that Clearwater simply passes through
- Requiring an external application server
- Not supported

Supported by Clearwater's Built-In MMTel Application Server

The following Supplementary Services are supported by Clearwater's built-in MMTel Application Server.

- Originating Identification Presentation - See the Privacy Feature document for more information.
- Originating Identification Restriction - See the Privacy Feature document for more information.
- Call Forwarding Unconditional - See the Call Diversion Support document for more information.
- Call Forwarding on not Logged in - See the Call Diversion Support document for more information.
- Call Forwarding on Busy - See the Call Diversion Support document for more information.
- Call Forwarding on not Reachable - See the Call Diversion Support document for more information.
- Call Forwarding on No Reply - See the Call Diversion Support document for more information.
- Barring of All Incoming Calls - See the Call Barring Support document for more information.
- Barring of All Outgoing Calls - See the Call Barring Support document for more information.
- Barring of Outgoing International Calls - See the Call Barring Support document for more information.

Supported by Clearwater Inherently

The following Supplementary Services are primarily implemented on the UE, and just require Clearwater to proxy messages and headers, which it does.

- Communication Hold
- Communication Waiting

Requiring an External Application Server

The following Supplementary Services require an external application server.

- Terminating Identification Presentation
- Terminating Identification Restriction
- Barring of Outgoing International Calls - ex Home Country - Clearwater does not currently have sufficient configuration to know the subscriber's home country.

- Message Waiting Indication - Clearwater proxies SUBSCRIBE and NOTIFY messages for the MWI event package (RFC 3842), but requires an external application server to handle/generate them.
- Ad-Hoc Multi Party Conference - Clearwater proxies messages between the UE and an external application server that conferences sessions together.

Not Supported

The following Supplementary Service is not currently supported by Clearwater.

- Barring of Incoming Calls - When Roaming - Clearwater does not currently support roaming, and so can't support barring on this basis.

Backups

Within a Clearwater deployment, Ellis and Vellum store persistent data (Bono, Sprout, Homer and Dime do not). To prevent data loss in disaster scenarios, Ellis and Vellum have data backup and restore mechanisms. Specifically, they support

- manual backup
- periodic automated local backup
- manual restore.

This document describes

- how to list the backups that have been taken
- how to take a manual backup
- the periodic automated local backup behavior
- how to restore from a backup.

Note that Vellum has 4 databases: * `homestead_provisioning` and `homestead_cache` for Homestead's data * `homer` for Homer's data * `memento` for Memento's data (if using the Memento AS)

Depending on your deployment scenario, you may not need to back up all of the data of Ellis and Vellum: * If your Clearwater deployment is integrated with an external HSS, the HSS is the master of Ellis' and some of Vellum's data, so you only need to backup/restore data in the `homer` and `memento` databases on Vellum * If you are not using a Memento AS, you do not need to backup/restore the `memento` database on Vellum

Listing Backups

The process for listing backups differs between Ellis and Vellum.

Ellis

To list the backups that have been taken on Ellis, run `sudo /usr/share/clearwater/ellis/backup/list_backups.sh`.

```
Backups for ellis:
1372294741  /usr/share/clearwater/ellis/backup/backups/1372294741
1372294681  /usr/share/clearwater/ellis/backup/backups/1372294681
```

```
1372294621  /usr/share/clearwater/ellis/backup/backups/1372294621
1372294561  /usr/share/clearwater/ellis/backup/backups/1372294561
```

Vellum

To list the backups that have been taken on Vellum, run

- `sudo /usr/share/clearwater/bin/list_backups.sh homestead_provisioning`
- `sudo /usr/share/clearwater/bin/list_backups.sh homestead_cache`
- `sudo /usr/share/clearwater/bin/list_backups.sh homer`
- `sudo /usr/share/clearwater/bin/list_backups.sh memento`

This produces output of the following form, listing each of the available backups.

```
No backup directory specified, defaulting to /usr/share/clearwater/homestead/backup/
↪backups
provisioning1372812963174
provisioning1372813022822
provisioning1372813082506
provisioning1372813143119
```

You can also specify a directory to search in for backups, e.g. for `homestead_provisioning`:

`sudo /usr/share/clearwater/bin/list_backups.sh homestead_provisioning <backup dir>`

Taking a Manual Backup

The process for taking a manual backup differs between Ellis and Vellum. Note that in both cases,

- the backup is stored locally and should be copied to a secure backup server to ensure resilience
- this process only backs up a single local node, so the same process must be run on all nodes in a cluster to ensure a complete set of backups
- these processes cause a small amount of extra load on the disk, so it is recommended not to perform this during periods of high load
- only 4 backups are stored locally - when a fifth backup is taken, the oldest is deleted.

Ellis

To take a manual backup on Ellis, run `sudo /usr/share/clearwater/ellis/backup/do_backup.sh`.

This produces output of the following form, reporting the successfully-created backup.

```
Creating backup in /usr/share/clearwater/ellis/backup/backups/1372336317/db_backup.sql
```

Make a note of the snapshot directory (`1372336317` in the example above) - this will be referred to as `<snapshot>` below.

This file is only accessible by the root user. To copy it to the current user's home directory, run

```
snapshot=<snapshot>
sudo bash -c 'cp /usr/share/clearwater/ellis/backup/backups/'$snapshot'/db_backup.sql␣
↪~'$USER' &&
              chown '$USER.$USER' db_backup.sql'
```

This file can, and should, be copied off the Ellis node to a secure backup server.

Vellum

To take a manual backup on Vellum, run

- `sudo cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh homestead_provisioning`
- `sudo cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh homestead_cache`
- `sudo cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh homer`
- `sudo cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh memento`

These each produce output of the following form, reporting the successfully-created backup.

```
...
Deleting old backup: /usr/share/clearwater/homestead/backup/backups/1372812963174
Creating backup for keyspace homestead_provisoning...
Requested snapshot for: homestead_provisioning
Snapshot directory: 1372850637124
Backups can be found at: /usr/share/clearwater/homestead/backup/backups/provisioning/
```

Note that Each of the Vellum databases will produce a different snapshot in a different directory.

The backups are only stored locally - the resulting backup for each command is stored in the listed directory. Make a note of the snapshot directory for each database - these will be referred to as `<snapshot>` below.

These should be copied off the node to a secure backup server. For example, from a remote location execute `scp -r ubuntu@<homestead node>:/usr/share/clearwater/homestead/backup/backups/provisioning/<snapshot> .`.

Periodic Automated Local Backups

Ellis and Vellum are automatically configured to take daily backups if you've installed them through chef, at midnight local time every night.

If you want to turn this on, edit your crontab by running `sudo crontab -e` and add the following lines if not already present:

- On Ellis:
  - `0 0 * * * /usr/share/clearwater/ellis/backup/do_backup.sh`
- On Vellum:
  - `0 0 * * * /usr/bin/cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh homestead_provisioning`

  - `5 0 * * * /usr/bin/cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh homestead_cache`
  - `10 0 * * * /usr/bin/cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh homer`
  - `15 0 * * * /usr/bin/cw-run_in_signaling_namespace /usr/share/clearwater/bin/do_backup.sh memento`

These backups are stored locally, in the same locations as they would be generated for a manual backup.

Restoring from a Backup

There are three stages to restoring from a backup.

1. Copying the backup files to the correct location.
2. Running the restore backup script.
3. Synchronizing Ellis' and Vellum's views of the system state.

This process will impact service and overwrite data in your database.

Copying Backup Files

The first step in restoring from a backup is getting the backup files/directories into the correct locations on the Ellis or Vellum node.

If you are restoring from a backup that was taken on the node on which you are restoring (and haven't moved it), you can just move onto the next step.

If not, create a directory on your system that you want to put your backups into (we'll use `~/backup` in this example). Then copy the backups there. For example, from a remote location that contains your backup directory `<snapshot>` execute `scp -r <snapshot> ubuntu@<vellum node>:backup/<snapshot>`.

On Ellis, run the following commands.

```
snapshot=<snapshot>
sudo chown root.root db_backup.sql
sudo mkdir -p /usr/share/clearwater/ellis/backup/backups/$snapshot
sudo mv ~/backup/$snapshot/db_backup.sql /usr/share/clearwater/ellis/backup/backups/
↪$snapshot
```

On Vellum there is no need to further move the files as the backup script takes a optional backup directory parameter.

If you are restoring a Vellum backup onto a completely clean deployment, you must ensure that the new deployment has at least as many Vellum nodes as the one from which the backup was taken. Each backup should be restored onto only one node, and each node should have only one backup restored onto it. If your new deployment does not have enough Vellum nodes, you should add more nodes and then, once restoring backups is complete, scale down your deployment to the desired size.

Running the Restore Backup Script

To actually restore from the backup file, run:

- On Ellis:
  - `sudo /usr/share/clearwater/ellis/backup/restore_backup.sh <snapshot>`

- On Vellum:
  - `sudo /usr/share/clearwater/bin/restore_backup.sh homestead_provisioning <hs-prov-snapshot> <backup directory>`
  - `sudo /usr/share/clearwater/bin/restore_backup.sh homestead_cache <hs-cache-snapshot> <backup directory>`
  - `sudo /usr/share/clearwater/bin/restore_backup.sh homer <homer-snapshot> <backup directory>`
  - `sudo /usr/share/clearwater/bin/restore_backup.sh memento <memento-snapshot> <backup directory>`

Note that, because the 4 Vellum databases are saved to different backups, the name of the snapshot used to restore each of the databases will be different.

Ellis will produce output of the following form.

```
Will attempt to backup from backup 1372336317
Found backup directory 1372336317
Restoring backup for ellis...
--------------
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */
--------------

...

--------------
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */
--------------
```

Vellum will produce output of the following form.

```
Will attempt to backup from backup 1372336442947
Will attempt to backup from directory /home/ubuntu/bkp_test/
Found backup directory /home/ubuntu/bkp_test//1372336442947
Restoring backup for keyspace homestead_provisioning...
xss =  -ea -javaagent:/usr/share/cassandra/lib/jamm-0.2.5.jar -
↪XX:+UseThreadPriorities -XX:ThreadPriorityPolicy=42 -Xm
s826M -Xmx826M -Xmn100M -XX:+HeapDumpOnOutOfMemoryError -Xss180k
Clearing commitlog...
filter_criteria: Deleting old .db files...
filter_criteria: Restoring from backup: 1372336442947
private_ids: Deleting old .db files...
private_ids: Restoring from backup: 1372336442947
public_ids: Deleting old .db files...
public_ids: Restoring from backup: 1372336442947
sip_digests: Deleting old .db files...
sip_digests: Restoring from backup: 1372336442947
```

For Vellum, after restoring the backups you must also do the following: - wait until the Cassandra process has restarted by running `sudo monit summary` and verifying that the `cassandra_process` is marked as `Running` - run `sudo cw-run_in_signaling_namespace nodetool repair`

At this point, this node has been restored.

Synchronization

It is possible (and likely) that when backups are taken on different boxes the data will be out of sync, e.g. Ellis will know about a subscriber, but there will no digest in Vellum. To restore the system to a consistent state we have a synchronization tool within Ellis, which can be run over a deployment to get the databases in sync. To run, log into an Ellis box and execute:

```
cd /usr/share/clearwater/ellis
sudo env/bin/python src/metaswitch/ellis/tools/sync_databases.py
```

This will:

- Run through all the lines on Ellis that have an owner and verify that there is a private identity associated with the public identity stored in Ellis. If successful, it will verify that a digest exists in Vellum for that private identity. If either of these checks fail, the line is considered lost and is removed from Ellis. If both checks pass, it will check that there is a valid iFC - if this is missing, it will be replaced with the default iFC.
- Run through all the lines on Ellis without an owner and make sure there is no orphaned data in Vellum, i.e. deleting the simservs, iFC and digest for those lines.

Shared Configuration

In addition to the data stored in Ellis and Vellum, a Clearwater deployment also has shared configuration that is automatically shared between nodes. This is stored in a distributed database, and mirrored to files on the disk of each node.

Backing Up

To backup the shared configuration:

- If you are in the middle of modifying shared config, complete the process to apply the config change to all nodes.
- Log onto one of the sprout nodes in the deployment.
- Copy the following files to somewhere else for safe keeping (e.g. another directory on the node, or another node entirely).

```
/etc/clearwater/shared_config
/etc/clearwater/bgcf.json
/etc/clearwater/enum.json
/etc/clearwater/s-cscf.json
/etc/clearwater/shared_ifcs.xml
/etc/clearwater/fallback_ifcs.xml
```

Restoring Configuration

To restore a previous backup, copy the six files listed above to `/etc/clearwater` on one of your sprout nodes. Then run the following commands on that node:

```
sudo cw-upload_shared_config
sudo cw-upload_bgcf_json
sudo cw-upload_enum_json
sudo cw-upload_scscf_json
```

```
sudo cw-upload_shared_ifcs_xml
sudo cw-upload_fallback_ifcs_xml
```

See Modifying Clearwater settings for more details on this.

Call Barring

Clearwater allows users to configure rules for blocking of incoming or outgoing calls, by implementing the IR.92 version of call barring (which is itself a cut-down version of the MMTEL call barring service).

The relevant specifications are:

- 3GPP TS 24.611 v11.2.0
  - The full specification
- GSMA PRD IR.92, Section 2.3.9
  - The required IR.92 subset

Some of the entries in the IR.92 list are not currently applicable for Clearwater, since Clearwater does not have support for/the concept of roaming calls. Because of this, the customer may only specify that they wish to block one of:

- No outgoing calls
- All outgoing calls
- Outgoing calls to international numbers (but see below)

And one of:

- No incoming calls
- All incoming calls

International Call detection

The 3GPP specification states that to detect an international number, a SIP router should discount SIP/SIPS URIs that do not have a `user=phone` parameter set on them (this parameter is intended to distinguish between dialed digits and call-by-name with numeric names) but X-Lite and other SIP clients never set this parameter. Because of this behavior, the Clearwater implementation considers all SIP URIs for international call categorization.

National dialing prefix

Another limitation is that the international call detection algorithm only checks against a global 'national' dialing prefix (which is fine unless subscribers are expected to be multi-national within one Clearwater deployment). There does not appear to be a mechanism in IMS networks to store a subscriber-specific locale (e.g. in the HSS) so this would require some smarts to solve. These issues may be addressed in subsequent development.

Call Diversion

Clearwater allows users to define rules for handling of incoming calls, by implementing a superset of the IR.92 version of call diversion (which is itself a cut-down version of the MMTEL call diversion service).

The relevant specifications are

- 3GPP TS 24.604
  - the full specification
- GSMA PRD IR.92, section 2.3.8
  - the required subset.

We have implemented all of IR.92 section 2.3.8. We have implemented all of 3GPP TS 24.604 with the following exceptions.

- CDIV Notification (notifying the diverting party when a call has been diverted) is not supported
- We only support the default subscription options from table 4.3.1.1 (most notably, no privacy options).
- We only support hard-coded values for network provider options from table 4.3.1.2, as follows.
  - We stop ringing the diverting party before/simultaneously with trying the diverted-to target (rather than waiting for them to start ringing).
    - Served user communication retention on invocation of diversion (forwarding or deflection) = Clear communication to the served user on invocation of call diversion
    - Served user communication retention when diverting is rejected at diverted-to user = No action at the diverting user
  - We don't support CDIV Notification (as above).
    - Subscription option is provided for "served user received reminder indication on outgoing communication that CDIV is currently activated" = No
    - CDIV Indication Timer = irrelevant
    - CDIVN Buffer Timer = irrelevant
  - We support up to 5 diversions and then reject the call.
    - Total number of call diversions for each communication = 5
    - AS behavior when the maximum number of diversions for a communication is reached = Reject the communication
  - The default no-reply timer before Communication Forwarding on no Reply occurs is hard-coded to 20s. This is still overridable on a per-subscriber basis.
    - Communication forwarding on no reply timer = 36s
- Obviously, the interactions with services that we haven't implemented yet have not been implemented! In particular, we haven't implemented any interactions with terminating privacy.
- We've ignored the comment in section 4.2.1.1 which says "It should be possible that a user has the option to restrict receiving communications that are forwarded" but doesn't provide any function for this.

Note that we *have* implemented support for Communication Deflection (in which the callee sends a 302 response and this triggers forwarding) despite it not being required by IR.92.

Elastic Scaling

The core Clearwater nodes have the ability to elastically scale; in other words, you can grow and shrink your deployment on demand, without disrupting calls or losing data.

This page explains how to use this elastic scaling function when using a deployment created through the automated or manual install processes. Note that, although the instructions differ between the automated and manual processes, the

underlying operations that will be performed on your deployment are the same - the automated process simply uses chef to drive this rather than issuing the commands manually.

Before scaling your deployment

Before scaling up or down, you should decide how many Bono, Sprout, Dime, Vellum and Homer nodes you need (i.e. your target size). This should be based on your call load profile and measurements of current systems, though based on experience we recommend scaling up a tier of a given type (sprout, bono, etc.) when the average CPU utilization within that tier reaches ~60%.

Performing the resize

If you did an Automated Install

To resize your automated deployment, run:

```
knife deployment resize -E <env> --bono-count <n> --sprout-count <n> --dime-count <n>␣
↪--vellum-count <n> --homer-count <n>
```

Where the <n> values are how many nodes of each type you need. Once this command has completed, the resize operation has completed and any nodes that are no longer needed will have been terminated.

More detailed documentation on the available Chef commands is available here.

If you did a Manual Install

If you're scaling up your deployment, follow the following process:

1. Spin up new nodes, following the standard install process, but with the following modifications:
   - Set the `etcd_cluster` so that it only includes nodes that are already in the deployment and that are running as etcd masters (so it does not include any nodes being added).
   - Stop when you get to the "Provide Shared Configuration" step. The nodes will learn their configuration from the existing nodes.
2. Wait until the new nodes have fully joined the existing deployment. To check if a node has joined the deployment:
   - Run `cw-check_cluster_state`. This should report that the cluster is stable and, if this is a Vellum node, it should also report that this node is in all of its clusters.
   - Run `sudo cw-check_config_sync`. This reports when the node has learned its configuration.
3. Update DNS to contain the new nodes.

If you're scaling down your deployment, follow the following process:

1. Update DNS to contain only the nodes that will remain after the scale-down.
2. On each node that is about to be turned down:
   - Run the appropriate command from the following (based on the node type) to stop processes from automatically restarting.
     - Bono - `monit unmonitor -g bono`
     - Sprout - `monit unmonitor -g sprout`

  - Dime - `monit unmonitor -g homestead && monit unmonitor -g homestead-prov && monit unmonitor -g ralf`
  - Vellum - n/a
  - Homer - `monit unmonitor -g homer`
  - Memento - `monit unmonitor -g sprout`
  - Ellis - `monit unmonitor -g ellis`
- Start the main processes quiescing.
  - Bono - `sudo service bono quiesce`
  - Sprout - `sudo service sprout quiesce`
  - Dime - `sudo service homestead stop && sudo service homestead-prov stop && sudo service ralf stop`
  - Vellum - n/a
  - Homer - `sudo service homer stop`
  - Memento - `sudo service sprout quiesce`
  - Ellis - `sudo service ellis stop`
- If the node you are turning down is a Vellum node, or another node configured to act as an etcd master, also do the following:
  - `sudo monit unmonitor -g clearwater_cluster_manager`
  - `sudo monit unmonitor -g clearwater_config_manager`
  - `sudo monit unmonitor -g clearwater_queue_manager`
  - `sudo monit unmonitor -g etcd`
  - `sudo service clearwater-etcd decommission`

  This will cause the node to leave its existing clusters.

3. Once the above steps have completed, turn down the nodes.

Privacy

The Clearwater privacy feature splits into two sections, configuration and application. For configuration, we've been working from a combination of 3GPP TS 24.623 and ETSI TS 129.364, which define the XDM documents that store the MMTel services configuration (currently only OIR, originating-identity-presentation-restriction, is supported). For application, we've used 3GPP TS 24.607‘ <http://www.quintillion.co.jp/3GPP/Specs/24607-910.pdf>‘__ which defines the roles of the originating UE, originating AS, terminating AS and terminating UE. This document is supplemented by RFC3323 and RFC3325.

For configuration, there's also a communication with Homestead to determine whether MMTel services should be applied for a call. This uses a proprietary (JSON) interface based on the inbound filter criteria definitions.

Configuration

User-specified configuration is held and validated by the XDM. When a subscriber is provisioned, Ellis injects a default document into the XDM for that user, this document turns off OIR by default. After this, if the user wishes to change their call services configuration, they use the XCAP interface (Ut) to change their document.

If the user has turned on Privacy by default, then all calls originating from the user will have identifying headers (e.g. `User-Agent`, `Organization`, `Server`...) stripped out from the originating request and will have the `From:` header re-written as `"Anonymous" <sip:anonymous@anonymous.invalid\>;tag=xxxxxxx`, preventing the callee from working out who the caller is. With the current implementation, there are some headers that are left in the message that could be used to determine some information about the caller, see below.

Application

Originating UE

The originating UE is responsible for adding any per-call Privacy headers (overriding the stored privacy configuration) if required and hiding some identifying information (e.g. `From:` header).

Application Server

When receiving a dialog-creating request from an endpoint, Sprout will ask Homestead which application servers should be used and will parse the returned iFC configuration. If the configuration indicates that MMTEL services should be applied, Sprout will continue to act as an originating and terminating application server on the call as it passes through.

Once Homestead has indicated that MMTel services should be applied for the call, Sprout will query the XDM to determine the user's specific configuration.

Originating Application Server

When acting as an originating AS, the above 3GPP spec states that we should do the following:

For an originating user that subscribes to the OIR service in "temporary mode" with default "restricted", if the request does not include a Privacy header field, or the request includes a Privacy header field that is not set to "none", the AS shall insert a Privacy header field set to "id" or "header" based on the subscription option. Additionally based on operator policy, the AS shall either modify the From header field to remove the identification information, or add a Privacy header field set to "user".

For an originating user that subscribes to the OIR service in "temporary mode" with default "not restricted", if the request includes a Privacy header field is set to "id" or "header", based on operator policy, the AS shall either, may modify the From header field to remove the identification information or add a Privacy header field set to "user".

Our implementation of Sprout satisfies the above by injecting the applicable Privacy headers (i.e. it never actually applies privacy at this point, only indicates that privacy should be applied later).

Terminating Application Server

When acting as a terminating AS, the 3GPP spec states that we should do the following:

If the request includes the Privacy header field set to "header" the AS shall:

a) anonymize the contents of all headers containing private information in accordance with IETF RFC 3323 [6] and IETF RFC 3325 [7]; and

b) add a Privacy header field with the priv-value set to "id" if not already present in the request.

If the request includes the Privacy header field set to "user" the AS shall remove or anonymize the contents of all "user configurable" headers in accordance with IETF RFC 3323 [6] and IETF RFC 3325 [7]. In the latter case, the AS may need to act as transparent back-to-back user agent as described in IETF RFC 3323 [6].

Our implementation of Sprout satisfies these requirements but does it by not fully satisfying the SHOULD clauses of the linked RFCs. In particular:

- 'header' privacy
  - We do not perform "Via-Stripping" since we use the Via headers for subsequent call routing and don't want to have to hold a via-header mapping in state.
  - Similarly we don't perform "Record-Route stripping" or "Contact-stripping".
- 'session' privacy
  - This is not supported so if the privacy is marked as critical, we will reject the call.
- 'user' privacy
  - We modify the To: header inline here (and do not cache the original value) we're assuming that the end-points are robust enough to continue correlating the call based on the tags
  - We do not anonymize the call-id (for the same reason as the Via-stripping above).
- 'id' privacy
  - This is fully supported.
- 'none' privacy
  - This is fully supported.

Terminating UE

The terminating UE may use the presence of Privacy headers to determine if privacy has been applied by OIR or by the originating endpoint (though this doesn't seem to be very useful information...). It may also look at the P-Asserted-Identity header to determine the true identity of the caller (rather than some local ID).

Dialog-Wide Issues

There are a few issues with our stateless model and privacy that are worth drawing out here, pending a fuller discussion.

- We only apply OIR processing and privacy obfuscation to originating requests (i.e. not on in-dialog requests or responses)
- We cannot apply Contact/Via/RR-stripping since to do so would require us to restore the headers on not just the response but also on all callee in-dialog requests.
- Similarly for Call-Id obfuscation.
- Stripping Contact/Via/RR would prevent features such as callee call push from working (since they require the callee to send a REFER that identifies the caller).

Off-net Calling with BGCF and IBCF

The Clearwater IBCF (Interconnection Border Control Function) provides an interface point between a Clearwater deployment and one or more trusted SIP trunks.

The IBCF does not currently provide all of the functions of an IMS compliant IBCF - in particular it does not support topology hiding or an H.248 interface to control a Transition Gateway.

To use the IBCF function you must install and configure at least one IBCF node, set up the ENUM configuration appropriately, and set up BGCF (Breakout Gateway Control Function) routing configuration on the Sprout nodes.

Install and Configure an IBCF

Install and configure an IBCF node with the following steps.

- Install the node as if installing a Bono node, either manually or using Chef. If using Chef, use the `ibcf` role, for example

```
knife box create -E <env> ibcf
```

- Edit the /etc/clearwater/user_settings file (creating it if it does not already exist) and add or update the line defining the IP addresses of SIP trunk peer nodes.

```
trusted_peers="<trunk 1 IP address>,<trunk 2 IP address>,<trunk 3 IP address>, ...
↪"
```

- Restart the Bono daemon.

```
service stop bono (allow monit to restart Bono)
```

ENUM Configuration

The number ranges that should be routed over the SIP trunk must be set up in the ENUM configuration to map to a SIP URI owned by the SIP trunk provider, and routable to that provider. For example, you would normally want to map a dialed number to a URI of the form `<number>@<domain>`.

This can either be achieved by defining rules for each number range you want to route over the trunk, for example

```
*.0.1.5 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@<trunk IP address>!" .
*.0.1.5.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@<trunk IP address>!" .
```

or by defining a default mapping to the trunk and exceptions for number ranges you want to keep on-net, for example

```
* IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@<trunk IP address>!" .
*.3.2.1.2.4.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@<local domain>!" .
*.3.2.1.2.4.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@<local domain>!" .
```

You can also use ENUM to provide number portability information, for example

```
*.3.2.1.2.4.2 IN NAPTR 1 1 "u" "E2U+pstn:tel" "!(^.*$)!sip:\\1;npdi;rn=+242123@<local␣
↪domain>!" .
*.3.2.1.2.4.2.1 IN NAPTR 1 1 "u" "E2U+pstn:tel" "!(^.*$)!sip:\\1;npdi@<local domain>!
↪" .
```

Refer to the ENUM guide for more about how to configure ENUM.

BGCF Configuration

BGCF (Border Gateway Control Function) configuration is stored in the `bgcf.json` file in `/etc/clearwater` on each Sprout node (both I- and S-CSCF). The `bgcf.json` file stores two types of mappings.

- The first maps from SIP trunk IP addresses and/or domain names to IBCF SIP URIs
- The second maps from a telephone number (contained in the `rn` parameter of a Tel URI, the `rn` parameter in a SIP URI, a TEL URI or the user part of a SIP URI with a user=phone parameter) to an IBCF SIP URI using prefix matching on the number.

These mappings control which IBCF nodes are used to route to a particular destination. Each entry can only apply to one type of mapping.

Multiple nodes to route to can be specified. In this case, Route headers are added to the message such that it is sent to the first node and the first node sends it to the second node and so on; the message is not tried at the second node if it fails at the first node.

The `bgcf.json` file is in JSON format, for example.

```
{
    "routes" : [
        {   "name" : "<route 1 descriptive name>",
            "domain" : "<SIP trunk IP address or domain name>",
            "route" : ["<IBCF SIP URI>"]
        },
        {   "name" : "<route 2 descriptive name>",
            "domain" : "<SIP trunk IP address or domain name>",
            "route" : ["<IBCF SIP URI>", "<IBCF SIP URI>"]
        },
        {   "name" : "<route 3 descriptive name>",
            "number" : "<Telephone number>",
            "route" : ["<IBCF SIP URI>", "<IBCF SIP URI>"]
        }
    ]
}
```

There can be only one route set for any given SIP trunk IP address or domain name. If there are multiple routes with the same destination IP address or domain name, only the first will be used. Likewise, there can only be one route set for any given telephone number; if there are multiple routes with the same telephone number only the first will be used.

A default route set can be configured by having an entry where the domain is set to `"*"`. This will be used by the BGCF if it is trying to route based on the the domain and there's no explicit match for the domain in the configuration, or if it is trying to route based on a telephone number and there's no explicit match for the number in the configuration.

After making a change to this file you should run `sudo cw-upload_bgcf_json` to ensure the change is synchronized to other Sprout nodes on your system (including nodes added in the future).

Multiple Domains

A single Clearwater deployment can support subscribers in multiple domains. This document

- gives some background context
- describes how to configure multiple domains
- covers some restrictions on this support.

Background

Clearwater acts as an S-CSCF for one or more home domains. It can only provide service to users within a home domain for which it is the S-CSCF. Traffic to or from users in any other home domains must go via the S-CSCF for those other domains.

Home domains are a routing concept. There is a similar (but distinct) concept of authentication realms. When a user authenticates, as well as providing their username and digest, they must also provide a realm, which describes which user database to authenticate against. Many clients default their home domain to match their authentication realm.

A single Clearwater deployment can support multiple home domains and multiple realms.

Configuration

There are three steps to configuring multiple home domains and/or multiple realms.

- Configure the Clearwater deployment to know about all the home domains it is responsible for.
- Add DNS records to point all home domains at the deployment.
- Create subscribers within these multiple home domains, and optionally using multiple different realms.

Configuring the Clearwater Deployment

The `/etc/clearwater/shared_config` file can contain two properties that are relevant to this function.

- `home_domain` - defines the "default" home domain and must always be set
- `additional_home_domains` - optionally defines additional home domains

The only difference between the default home domain and the additional home domains is that the default home domain is used whenever Clearwater must guess the home domain for a TEL URI, e.g. when filling in the orig-ioi or term-ioi parameters on the P-Charging-Vector header. This is fairly minor, and generally you can consider the default home domain and additional home domains as equivalent.

Adding DNS Records

The Clearwater DNS Usage document describes how to configure DNS for a simple single-domain deployment. This includes creating NAPTR, SRV and A records that resolve the home domain (or `<zone>`) to the bono nodes. If you have additional home domains, you should repeat this for each additional home domain. This ensures that SIP clients can find the bono nodes without needing to be explicitly configured with a proxy.

Creating subscribers.

Once the deployment has multiple domain support enabled, you can create subscribers in any of the domains, and with any realm.

- If you use ellis, the `` `create-numbers.py <https://github.com/Metaswitch/ellis/blob/dev/docs/create-numbers.md>`__ `` script accepts a `--realm` parameter to specify the realm in which the directory numbers are created. When a number is allocated in the ellis UI, ellis picks any valid number/realm combination.
- If you bulk-provision numbers, the home domain and realm can be specified in the input CSV file.
- If you use homestead's provisioning API (i.e. without an external HSS), you can specify the home domain and realm as parameters.

Alternatively, you can use an external HSS, and configure the subscribers using its provisioning interface.

Restriction

Clearwater does not support connections to multiple Diameter realms for the Cx interface to the HSS or the Rf interface to the CDF - only a single Diameter realm is supported.

Multiple Network Support

Introduction

As of the Ultima release, Project Clearwater can be configured to provide signaling service (SIP/HTTP/Diameter) on a separate physical (or virtual) network to management services (SNMP/SSH/Provisioning). This may be used to protect management devices (on a firewalled network) from attack from the outside world (on the signaling network).

Technical Details

Network Independence

If traffic separation is configured, the management and signaling networks are accessed by the VM via completely separate (virtual) network interfaces.

The management and signaling networks may be completely independent.

- There need not be any way to route between them.
- The IP addresses, subnets and default gateways used on the networks may be the same, may overlap or may be completely distinct.

Traffic Distinction

The following traffic is carried over the management interface:

- SSH
- DNS (for management traffic)
- SNMP statistics
- SAS
- NTP
- Homestead provisioning
- Homer provisioning
- Ellis provisioning API
- M/Monit

The following traffic is carried over the signaling interface:

- DNS (for signaling traffic)
- ENUM
- SIP between the UE, Bono and Sprout
- Cx between Dime and the HSS
- Rf between Dime and the CDF
- Ut between Homer and the UE
- inter-tier HTTP
- intra-tier memcached, Cassandra and Chronos flows

Both interfaces respond to ICMP pings which can be used by external systems to perform health-checking if desired.

Configuration

Pre-requisites

The (possibly virtual) machines that are running your Project Clearwater services will need to be set up with a second network interface (referred to as `eth1` in the following).

You will need to determine the IP configuration for your second network, including IP address, subnet, default gateway, DNS server. DHCP is supported on this interface if required but you should ensure that the DHCP lease time is sufficiently high to ensure that the local IP address does not change regularly.

Preparing the network

Due to the varied complexities of IP networking, it would be impractical to attempt to automate configuring the Linux-level view of the various networks. Network namespaces are created and managed using the `ip netns` tool, which is a standard part of Ubuntu 14.04.

The following example commands (when run as root) create a network namespace, move `eth1` into it, configure a static IP address and configure routing.

```
ip netns add signaling
ip link set eth1 netns signaling
ip netns exec signaling ifconfig lo up
ip netns exec signaling ifconfig eth1 1.2.3.4/16 up
ip netns exec signaling route add default gateway 1.2.0.1 dev eth1
```

Obviously you should make any appropriate changes to the above to correctly configure your chosen signaling network. These changes are **not** persisted across reboots on Linux and you should ensure that these are run on boot before the `clearwater-infrastructure` service is run. A sensible place to configure this would be in `/etc/rc.local`.

Finally, you should create `/etc/netns/signaling/resolv.conf` configuring the DNS server you'd like to use on the signaling network. The format of this file is documented at http://manpages.ubuntu.com/manpages/trusty/man5/resolv.conf.5.html but a simple example file might just contain the following.

```
nameserver <DNS IP address>
```

Project Clearwater Configuration

Now that the signaling namespace is configured and has networking set up, it's time to apply it to Project Clearwater. To do this, change the `local_ip`, `public_ip` and `public_hostname` lines in `/etc/clearwater/local_config` to refer to the node's identities in the signaling network and add the following lines:

```
signaling_namespace=<namespace name>
signaling_dns_server=<DNS IP address>
management_local_ip=<Node's local IP address in the management
```

If you've not yet installed the Project Clearwater services, do so now and the namespaces will be automatically used.

If you've already installed the Project Clearwater services, run `sudo service clearwater-infrastructure restart` and restart all the Clearwater-related services running on each machine (you can see what these are with `sudo monit summary`).

Diagnostic Changes

All the built-in Clearwater diagnostics will automatically take note of network namespaces, but if you are running diagnostics yourself (e.g. following instructions from the troubleshooting page) you may need to prefix your commands with `ip netns exec <namespace>` to run them in the signaling namespace. The following tools will need this prefix:

- `cqlsh` - For viewing Cassandra databases
- `nodetool` - For viewing Cassandra status
- `telnet` - For inspecting Memcached stores

Geographic redundancy

This article describes the architecture of a geographically-redundant deployment in Clearwater.

Details on how to configure a geographically redundant deployment are available here, and how to recover a deployment after site failure is described here.

Architecture

The architecture of a geographically-redundant system is as follows.

Each site has its own, separate, etcd cluster. This means that Clearwater's automatic configuration sharing only applies within a site, not between sites. Data is shared between the sites primarily by Vellum.

Vellum has 3 databases, which support Geographic Redundancy differently:

- The Homestead-Prov, Homer and Memento databases are backed by Cassandra, which is aware of local and remote peers, so these are a single cluster split across the two geographic regions.
- Chronos is aware of local peers and the remote cluster, and handles replicating timers across the two sites itself.
- There is one memcached cluster per geographic region. Although memcached itself does not support the concept of local and remote peers, Vellum runs Astaire as a memcached proxy which allows Sprout and Dime nodes to build geographic redundancy on top - writing to both local and remote clusters, and reading from the local but falling back to the remote.

Sprout nodes use the local Vellum cluster for Chronos and both local and remote Vellum clusters for memcached (via Astaire). If the Sprout node includes Memento, then it also uses the local Vellum cluster for Cassandra. Dime nodes use the local Vellum cluster for Chronos and both local and remote Vellum clusters for memcached (via Astaire). If Homestead-Prov is in use, then it also uses the local Vellum cluster for Cassandra.

Communications between nodes in different sites should be secure - for example, if it is going over the public internet rather than a private connection between datacenters, it should be encrypted and authenticated with (something like) IPsec.

Ellis is not redundant, whether deployed in a single geographic region or more. It is deployed in one of the geographic regions and a failure of that region would deny all provisioning function

Separate instances of Bono in each geographic region front the Sprouts in that region. The Bonos should be able to contact the Sprouts in either region.

The architecture above is for 2 geographic regions - Project Clearwater does not currently support more regions.

Note that there are other servers involved in a deployment that are not described above. Specifically,

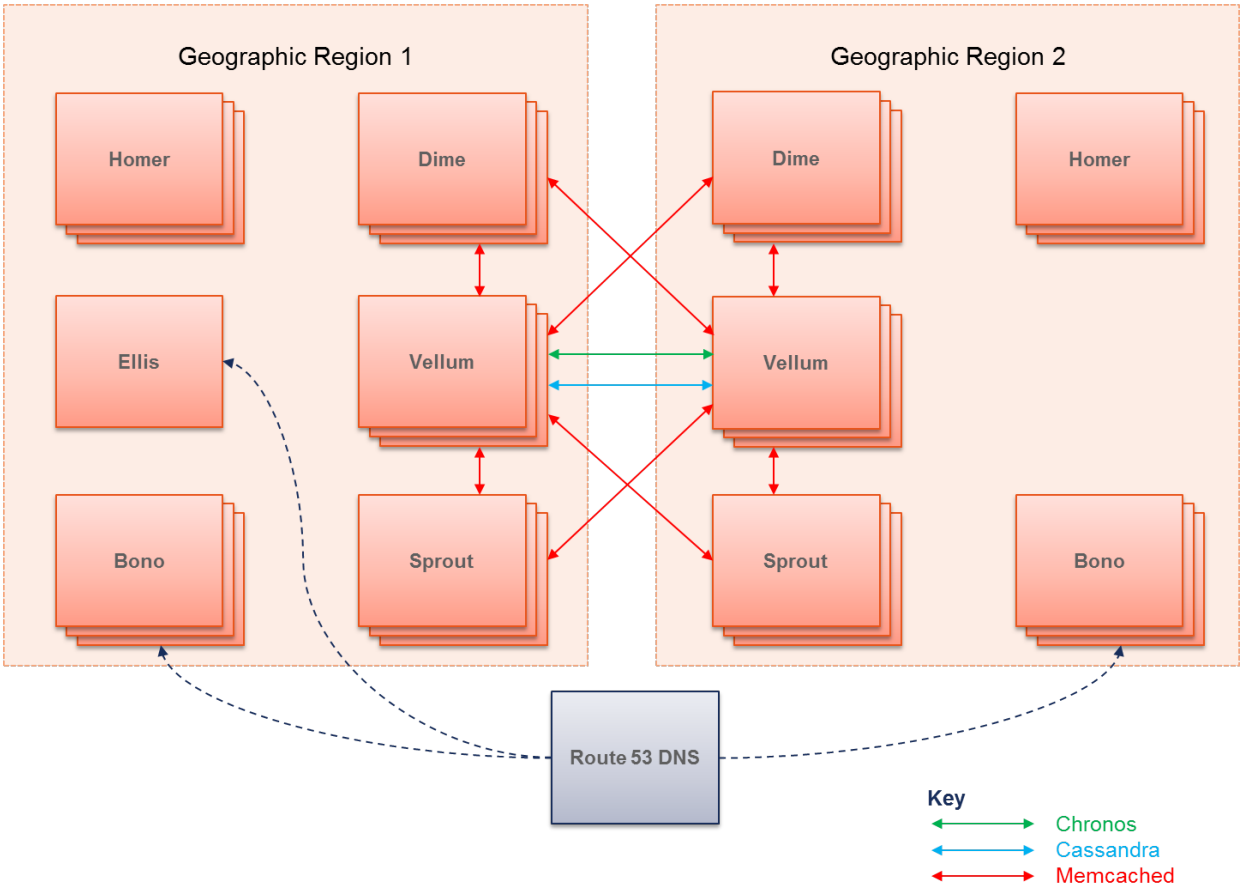

Fig. 8.2: Simplified GR Architecture

- Communication back to the repository server is via HTTP. There is a single repository server. The repository server is not required in normal operation, only for upgrades.

Limitations

The local IP addresses of all nodes in a deployment most be reachable from all other nodes - there must not be a NAT between the two GR sites. (This currently precludes having the GR sites in different EC2 regions.)

IPv6

IPv6 is the new version of IP. The most obvious difference from IPv4 is the addresses.

- IPv4 addresses are 4 bytes, represented in decimal, separated by dots, e.g. `166.78.3.224`
- IPv6 addresses are 16 bytes, represented in hexadecimal, with each pair of bytes separated by colons, e.g `2001:0db8:85a3:0000:0000:8a2e:0370:7334`.

Clearwater can operate over either IPv4 or IPv6, but not both simultaneously (known as "dual-stack").

This document describes

- how to configure Clearwater to use IPv6
- function currently missing on IPv6.

Configuration

As discussed in the Clearwater installation instructions, there are several ways to install Clearwater, and which way you choose affects how you must configure IPv6.

Manual Install

The process to configure Clearwater for IPv6 is very similar to IPv4. The key difference is to use IPv6 addresses in the `/etc/clearwater/local_config` file rather than IPv4 addresses when following the manual install instructions.

Note also that you must configure your DNS server to return IPv6 addresses (AAAA records) rather than (or as well as) IPv4 addresses (A records). For more information on this, see the Clearwater DNS usage documentation.

Automated Install

Configuring Clearwater for IPv6 via the automated install process is not yet supported (and Amazon EC2 does not yet support IPv6).

All-in-one Images

Clearwater all-in-one images support IPv6.

Since all-in-one images get their IP configuration via DHCP, the DHCP server must be capable of returning IPv6 addresses. Not all virtualization platforms support this, but we have successfully tested on OpenStack.

Once the all-in-one image determines it has an IPv6 address, it automatically configures itself to use it. Note that since Clearwater does not support dual-stack, all-in-one images default to using their IPv4 address in preference to their

IPv6 address if they have both. You should make sure your DHCP server only offers IPv6 addresses to the all-in-one image if you want to use DHCP.

Missing Function

The following function is not yet supported on IPv6.

- Automated install
- SIP/WebSockets (WebRTC)

SIP Interface Specifications

This document is a review of all the SIP related RFCs referenced by IMS (in particular TS 24.229 Rel 10), particularly how they relate to Clearwater's SIP support. The bulk of the document goes through the RFCs grouped in the following 4 categories.

- RFCs already supported by Clearwater (at least sufficient for IMS compliance for Clearwater as a combined P/I/S-CSCF/BGCF/IBCF).
- RFCs partially supported by Clearwater.
- RFCs that are relevant to Clearwater but not currently supported.
- RFCs relevant to media processing, which Clearwater doesn't currently need to handle as it is not involved in the media path.
- RFCs that are not relevant to Clearwater.

Supported

The following RFCs are already supported by Clearwater. Note that a number of these are supported simply because they require no active function from SIP proxies other than forwarding headers unchanged.

Basic SIP (RFC 3261)

- This covers the basic SIP protocol including
  - the 6 basic methods (ACK, BYE, CANCEL, INVITE, OPTIONS and REGISTER)
  - the 44 basic headers
  - the transaction and dialog state machines
  - the function of UACs, UASs, stateful and stateless proxies, and registrars
  - basic and digest security
  - transport over UDP and TCP.
- Clearwater supports this RFC.

SUBSCRIBE/NOTIFY/PUBLISH messages (RFC 3265 and RFC 3903)

- These two RFCs define a framework for generating and distributing event notifications in a SIP network. They cover the SUBSCRIBE and NOTIFY methods and Allow-Events and Event headers (RFC 3265) and the PUBLISH method and SIP-Etag and SIP-If-Match headers (RFC 3903).
- Routing of these messages is largely transparent to proxies, so is largely already supported by proxy function in Clearwater.

UPDATE method (RFC 3311)

- Defines UPDATE method used to update session parameters (so can be used as an alternative to reINVITE as a keepalive, for media renegotiation or to signal successful resource reservation). Often used by ICE enabled endpoints to change media path once ICE probing phase has completed.
- Supported transparently by Clearwater.

Privacy header (RFC 3323)

- Required for privacy service.
- Clearwater supports this header.

P-Asserted-Identity and P-Preferred-Identity headers (RFC 3325)

- Defines headers that allow a UE to select which identity it wants to use on a call-by-call basis. Ties in with authentication and the P-Associated-URIs header defined in RFC 3455 (see below).
- Clearwater supports these headers.

Path header (RFC 3327)

- Defines handling of registrations from devices which are not adjacent (in SIP routing terms) to the registrar.
- Supported by Clearwater.

message/sipfrag MIME type (RFC 3420)

- RFC 3420 defines an encoding for transporting MIME or S/MIME encoded SIP message fragments in the body of a SIP message. Use cases for this include status NOTIFYs sent during REFER processing which signal the status of the referral to the referee.
- Support for this is mandatory according to TS 24.229, but it does not describe any particular use cases.
- Transparent to proxy components, so supported by Clearwater.

P-Access-Network-Info, P-Associated-URI, P-Called-Party-ID, P-Charging-Function-Address, P-Charging-Vector and P-Visited-Network-ID headers (RFC 3455)

- Defines various private headers specifically for IMS.

- P-Access-Network-Info is added to incoming messages by the UE or the P-CSCF to provide information about the access network and possibly the UEs location within the network (for example cell). IMS specifications talk about various uses for this information, including routing emergency calls, an alternative to `phone-context` for interpreting dialled digits, determining the security scheme. This header is supported by Clearwater.
- P-Associated-URI is returned by registrar to include the list of alternative user identities defined for the IMS subscription. This header is supported by Clearwater.
- P-Called-Party-ID is added to a SIP request by an IMS network to ensure the called UE knows which of the user identities within the IMS subscription was actually called. This header is supported by Clearwater.
- P-Charging-Function-Address and P-Charging-Vector headers are related to billing, and are supported by Clearwater.
- P-Visited-Network-ID is used when roaming to identify the network the user has roamed to. The specs say it is a free-form string, and should be used to check that there is a roaming agreement in place. This header is supported by Clearwater.

Symmetric SIP response routing (RFC 3581)

- Defines how SIP responses should be routed to ensure they traverse NAT pinholes.
- Mandatory for IMS proxy components.
- Supported by Clearwater nodes.

Call transfer, multiparty call control (REFER method and related headers) (RFC 3515, RFC 3891, RFC 3892, RFC 3911)

- Covers REFER method, Refer-To header and event packages for transferring referral state (RFC 3515), Replaces header (RFC 3891), Referred-By header (RFC 3892), Join header (RFC 3911), Refer-Sub header (RFC 4488), and the Target-Dialog header (RFC 4538).
- Transparent to proxies, so supported by Clearwater - although needs support for GRUUs (introduced in the Ninja Gaiden release) to work in all use cases.

Service-Route header (RFC 3608)

- In IMS an S-CSCF includes Service-Route header on REGISTER responses, so UE can use to construct Route headers to ensure subsequent requests get to the appropriate S-CSCF.
- Optional according to TS 24.229(http://www.3gpp.org/ftp/Specs/html-info/24229.htm).
- This header is supported by Clearwater.

ENUM (RFC 3761 and RFC 4769)

- Defines a mechanism for using DNS look-ups to translate telephone numbers into SIP URIs. (RFC 3761 defines ENUM, RFC 4769 contains IANA registrations for ENUM data.)
- IMS specifies that ENUM is one mechanisms an S-CSCF can use to translate non-routable URIs (either Tel URIs or SIP URIs encoding a phone number) into a globally routable URI (one where the domain name/IP address portion of the URI can safely be used to route the message towards its destination).
- Clearwater supports this through function in Sprout.

Event package for MWI (RFC 3842)

- Defines an event package for notifying UAs of message waiting.
- Required for UEs and ASs implementing MWI services, transparent to proxy components.
- Transparent to proxies, so already supported by Clearwater.

Presence event package (RFC 3856)

- Defines an event package for monitoring user presence.
- Required for UEs supporting presence and presence servers.
- Transparent to proxies, so supported by Clearwater.

Watcher event template package (RFC 3857)

- Defines an event package that can be used to subscribe to the list of watchers for another event type.
- IMS mandates support for this package for subscribing to the list of watchers of a particular presentity, so support is only applicable for SIP UEs supporting presence and presence application servers.
- Transparent to proxies, so supported by Clearwater.

Session-expiry (RFC 4028)

- Covers periodic reINVITE or UPDATE messages used to allow UEs or call stateful proxies to detect when sessions have ended. Includes Min-SE and Session-Expires headers, which are used to negotiate the frequency of keepalives.
- Clearwater supports session-expiry as a proxy per section 8 of the RFC. It does not have a minimum acceptable session interval, so does not make use of the Min-SE header, though does honor it.
- Clearwater requests a session interval of 10 minutes (or as close to 10 minutes as possible given pre-existing headers) in order to properly perform session-based billing.

Early session disposition type (RFC 3959)

- Defines a new value for the Content-Disposition header to indicate when SDP negotiation applies to early media.
- Optional according to TS 24.229.
- Transparent to proxies so supported by Clearwater.

Dialog events (RFC 4235)

- Event package for dialog state.
- In TS 24.229 only seems to be relevant to E-CSCF and LRF functions.
- Transparent to proxies, so supported by Clearwater.

MRFC control (RFC 4240, RFC 5552, RFC 6230, RFC 6231, RFC 6505)

- These RFCs define three different ways of controlling the function of an MRFC from an AS. RFC 4240 is a simple "play an announcement" service, RFC 5552 uses VoiceXML and RFC 6230/RFC 6231/RFC 6505 use SIP/SDP to establish a two-way control channel between the AS and MRFC.
- IMS allows any of the three mechanisms to be used, or combinations depending on circumstances.
- All three mechanisms are transparent to proxy components, so supported by Clearwater.

History-Info (draft-ietf-sipcore-rfc4244bis - not RFC 4244, which is inaccurate wrt real implementations)

- Primarily aimed at ASs - need to manipulate History-Info headers when diverting calls. The MMTEL AS built into Sprout already supports this for CDIV.
- Also, need to proxy History-Info headers transparently. Clearwater supports this.
- However, s9.1 says if the incoming H-I is missing or wrong, the intermediary must add an entry on behalf of the previous entity. Clearwater does not currently do this so if other parts of the network are not compliant, Clearwater will not fill in for them (and should).

OMS Push-to-Talk over Cellular service (RFC 4354, RFC 4964 and RFC 5318)

- Covers poc-settings event package (RFC 4354), P-Answer-State header (RFC 4964) and P-Refused-URI-List header (RFC 5318)
- Only required for OMA push-to-talk over cellular service.
- Optional according to TS 24.229.
- Transparent to proxies, so supported by Clearwater.

Conference event package (RFC 4575)

- Defines an event package for various events associated with SIP conferences.
- Mandatory for UEs and IMS application servers providing conference services.
- Transparent to proxies, so supported by Clearwater.

Event notification resource lists (RFC 4662)

- Defines an extension to the SUBSCRIBE/NOTIFY event mechanisms that allow an application to subscribe for events from multiple resources with a single request, by referencing a resource list.
- In IMS, this is only required for presence events, and is only mandatory for UEs supporting presence or presence servers, otherwise it is optional.
- Should be transparent to proxies, so supported by Clearwater.

Consent based Communications (RFC 5360)

- Defines a framework for obtaining consent for routing traffic towards a SIP entity, including Permission-Missing and Trigger-Consent headers, SIP MESSAGE methods sent via store-and-forward relay to request consents and an event package (ie. a lot of extra SIP!).
- In theory could be applicable to registrations in IMS (guarding against a user maliciously redirecting their incoming calls or messages to another user's device) or to CDIV type services.
- Mandatory according to TS 24.229 - but not clear exactly what is required, could be as simple as just forwarding headers transparently, or it could be required to support consent for registrations.
- Reading Rel 9, this strongly suggests that the only nodes that need to actually invoke these mechanisms certain types of application servers, such as conference servers or message servers that support message distribution lists (so the function is to guard against people being added to these lists without their consent). Therefore the only function required in the core is to pass the messages and headers through transparently. Clearwater already supports this minimal set of function.

Multiple-Recipient MESSAGE requests (RFC 5365)

- Defines a mechanism for MESSAGE requests to be sent to multiple recipients by specifying the list of recipients in the body of the message. A message list server then fans the MESSAGE out to the multiple recipients.
- The function is mandatory in an IMS message list server, but not required anywhere else in the network.
- Transparent to proxies, so supported by Clearwater.

Creating multi-party conferences (RFC 5366)

- Allows a SIP UA to establish a multi-party conference by specifying a resource list in the body of the message used to set up the conference.
- Mandatory for IMS conference app servers.
- Transparent to proxies, so supported by Clearwater.

Referring to multiple resources (RFC 5368)

- Allows a SIP UA to send a REFER message containing a resource list URI. One use case for this could be to send a single REFER to a conference server to get it to invite a group of people to a conference.
- Optional according to TS 24.229, and only applicable to UEs and any application servers that can receive REFER requests.
- Transparent to proxies, so supported by Clearwater.

P-Served-User header (RFC 5502)

- Only applicable on the ISC interface - used to clear up some ambiguities about exactly which user the AS should be providing service for.
- Optional according to TS 24.229.
- This header is supported by Clearwater and included on requests sent to application servers.

Message body handling (RFC 5621)

- Defines how multiple message bodies can be encoded in a SIP message.
- Relevant in handling of third-party REGISTERs on ISC where XML encoded data from iFC may be passed to AS.

SIP outbound support (RFC 5626)

- Defines mechanisms for clients behind NATs to connect to a SIP network so SIP requests can be routed to the client through the NAT.
- Mandatory according to TS 24.229.
- Supported by Clearwater for NAT traversal, except for the Flow-Timer header (which tells the client how often to send keepalives).

Fixes to Record-Route processing (RFC 5658)

- Fixes some interoperability issues in basic SIP handling of Record-Route headers, particularly in proxies which have multiple IP addresses.
- Clearwater used RFC5658 double-record routing in Bono nodes when transitioning between the trusted and untrusted zones on different port numbers.

Fixes for IPv6 addresses in URIs (RFC 5954)

- Fixes a minor problem in the ABNF definition in RFC 3261 relating to IPv6 addresses, and clarifies URI comparison rules when comparing IPv6 addresses (in what looks like an obvious way).
- Mandatory according to TS 24.229.
- Supported by Clearwater.

Q.950 codes in Reason header (RFC 6432)

- Defines how Q.950 codes can be encoded in Reason header.
- Mandatory for MGCFs, optional for proxy components in IMS.
- Transparent to proxies anyway, so supported by Clearwater.

Geolocation (RFC 4483 and RFC 6442)

- Framework for passing geo-location information within SIP messages.
- According to TS 24.229 only required on an E-CSCF.
- Supported by Clearwater as headers will be passed transparently.

Proxy Feature Capabilities (referenced as draft-ietf-sipcore-proxy-feature-12, but now RFC 6809)

- Defines a mechanism to allow SIP intermediaries (such as registrars, proxies or B2BUAs) to signal feature capabilities when it would not be appropriate to use the feature tags mechanism in the Contact header as per RFC 3841.
- Mandatory on P-CSCF according to TS 24.229, but not clear what this actually means as the IMS specs don't seem to actually define any features to be signalled in these headers.
- Clearwater will forward these headers if specified by other nodes in the signaling path, so this is supported.

Alert info URNs (draft-ietf-salud-alert-info-urns-06)

- Defines family of URNs to be used in Alert-Info header to provide better control over how user is alerted on a UE.
- Optional according to TS 24.229.
- Transparent to proxies, so supported by Clearwater.

AKA Authentication (RFC 3310)

- This RFC defines how AKA authentication parameters map into the authentication and authorization headers used for digest authentication.
- IMS allows AKA authentication as an alternative to SIP digest, although it is not mandatory.
- While Bono doesn't support AKA, the IMS Core part of Clearwater (Sprout, Vellum and Dime) has support for both AKAv1 and AKAv2.

SIP Instant Messaging (RFC 3428)

- Defines the use of the SIP MESSAGE method to implement an instant messaging service.
- Mandatory in proxy components according to TS 24.229.
- Supported in Clearwater.

Registration Events (RFC 3680)

- Defines an event package supported by SIP registrars to notify other devices of registrations.
- Must be supported by IMS core for the ISC interface, and also needed by some UEs.
- Supported in Clearwater since sprint 40 "Yojimbo"

PRACK support (RFC 3262)

- Defines a mechanism for ensuring the reliability of provisional responses when using an unreliable transport. Covers the PRACK method and the Rseq and Rack headers.
- Optional according to TS 24.229.
- Supported in Clearwater.

Fixes to issues with SIP non-INVITE transactions (RFC 4320)

- Defines changes to RFC 3261 procedures for handling non-INVITE transactions to avoid some issues - in particular the potential for an O(N^2) storm of 408 responses if a transaction times out.
- Mandatory for all SIP nodes according to TS 24.229.
- Supported in Clearwater.

User agent capabilities and caller preferences (RFC 3840 and RFC 3841)

- User agent capabilities encoded as feature tags in Contact headers during registration (RFC 3840) and Accept-Contact, Reject-Contact and Request-Disposition headers encode filtering rules to decide which targets subsequent request should be routed/forked to (RFC 3841).
- Used for routing of requests to targets with the appropriate features/capabilities in IMS. Mandatory for proxy components.
- Clearwater's Sprout registrar has full support for storing, matching, filtering and prioritizing bindings based on advertised capabilities and requirements as per these specifications.

AKAv2 (RFC 4169)

- An updated version of Authentication and Key Agreement, which incorporates the integrity key and cryptographic key into the response calculation.
- While Bono doesn't support AKA, the IMS Core part of Clearwater (Sprout, Vellum and Dime) has support for both AKAv1 and AKAv2.

P-Profile-Key header (RFC 5002)

- Used solely between I-CSCF and S-CSCF to signal the public service identity key to be used when a requested public service identity matches a wildcard entry in the HSS. Purely an optimization to avoid having to do wildcard matching twice for a single request.
- Optional according to TS 24.229
- Supported in Clearwater.

Relevant to Clearwater and partially supported

GRUUs (RFC 5627, plus RFC 4122, draft-montemurro-gsma-imei-urn-11 and draft-atarius-dispatch-id-meid-urn-10)

- RFC 5627 defines mechanisms to assign and propagate a Globally-Routable User Agent URI for each client that registers with the SIP network. A GRUU identifies a specific user agent rather than a SIP user, and is routable from anywhere in the internet. GRUUs are intended to be used in scenarios like call transfer where URIs are required for individual user agents to ensure the service operates correctly. Standard Contact headers would seem to do the job in many cases but don't satisfy the globally routable requirement in all cases, for example where the client is behind certain types of NAT.
- Assignment and use of GRUUs is mandatory for S-CSCF and UEs in an IMS network. RFC 4122, draft-montemurro-gsma-imei-urn-11 and draft-atarius-dispatch-id-meid-urn-10 are referenced from the sections of TS 24.229 that discuss exactly how GRUUs should be constructed.

- Clearwater supports assigning and routing on public GRUUs as of the Ninja Gaiden release. It does not support temporary GRUUs.
- The Blink softphone client can be used to test this, as it provides the +sip.instance parameter needed for GRUU support.

Registration event package for GRUUs (RFC 5628)

- Defines an extension to the RFC 3680 registration event package to include GRUUs.
- Mandatory on S-CSCF and UEs in an IMS network.
- Clearwater includes public GRUUs in these event notifications as of the Ninja Gaiden release. It does not support temporary GRUUs.

Alternative URIs (RFC 2806, RFC 2368, RFC 3859, RFC 3860, RFC 3966)

- Various RFCs defining alternatives to SIP URI - Tel URI (RFC 2806 and RFC 3966), mailto URI (RFC 2368), pres URI (RFC 3859), and im URI (RFC 3860).
- IMS allows use of Tel URIs
  - as a public user identity associated with a subscription (although a subscription must have at least on public user identity which is a SIP URI)
  - as the target URI for a call.
- Other URIs can be specified as Request URI for a SIP message.
- Clearwater supports SIP and Tel URIs but not mailto, pres or im URIs.

Number portability parameters in Tel URI (RFC 4694)

- Defines additional parameters in Tel URI for signaling related to number portability.
- Used in IMS in both Tel and SIP URIs for carrier subscription scenarios, and in IMS core for other number portability related scenarios.
- Optional according to TS 24.229.
- The `rn` and `npdi` parameters are supported by Clearwater; Clearwater doesn't currently support the other parameters defined in this RFC.

Relevant to Clearwater but not currently supported

These are the RFCs which are relevant to Clearwater and not yet supported.

Dialstring URI parameter (RFC 4967)

- Defines a `user=dialstring` parameter used in SIP URIs to indicate that the user portion of the URI is a dial string (as opposed to a number that definitely identifies a phone as in the `user=phone` case).
- IMS allows this encoding from UEs initiating calls, but doesn't specify any particular processing within the core of the network. The intention is that this can be handled by an application server, or captured by filter criteria.
- Clearwater doesn't currently support this.

P-Early-Media header (RFC 5009)

- Used to authorize and control early media.
- If P-CSCF is not gating media then required function is as simple as
  - adding P-Early-Media header with `supported` value on requests from clients (or modifying header from clients if already in message)
  - passing the header through transparently on responses.
- If P-CSCF is gating media then function is more complex as P-CSCF has to operate on values in P-Early-Media headers sent to/from UEs.
- Mandatory in a P-CSCF according to TS 24.229.
- Clearwater doesn't currently support this.

P-Media-Authorization header (RFC 3313)

- According to TS 24.229, only required if P-CSCF supporting IMS AKA authentication with IPsec ESP encryption, or SIP digest authentication with TLS encryption.
- Not supported, as this is P-CSCF only (and Bono doesn't support AKA).

Signalling Compression aka SigComp (RFC 3320, RFC 3485, RFC 3486, RFC 4077, RFC 4896, RFC 5049, RFC 5112)

- RFC 3320 defines basic SigComp (which is not SIP-specific), RFC 3485 defines a static dictionary for use in SigComp compression of SIP and SDP, RFC 3486 defines how to use SigComp with SIP, RFC 4077 defines a mechanism for negative acknowledgements to signal errors in synchronization between the compressor and decompressor, RFC 4896 contains corrections and clarifications to RFC 3320, RFC 5049 contains details and clarifications for SigComp compression of SIP, and RFC 5112 defines a static dictionary for use in SigComp compression of SIP presence bodies.
- TS 24.229 says that SigComp is mandatory between UEs and P-CSCF if access network is one of the 3GPP or 802.11 types (specifically it says this is mandatory if the UE sends messages with the P-Access-Network-Info set to one of these values - ie. the UE knows that is the type of access network it is being used on). Compression must use the dictionaries defined in both RFC 3485 and RFC 5112.
- Clearwater does not currently support SigComp (although it would be relatively straightforward to implement it).

Reason header (RFC 3326)

- Already supported in Clearwater for responses, would need to add support for passing on CANCEL requests (but pretty easy).
- Optional according to TS 24.229.

Security-Client, Security-Server and Security-Verify headers (RFC 3329)

- Defines headers that can be used to negotiate authentication mechanisms.
- Only required if P-CSCF supporting IMS AKA authentication with IPsec ESP encryption, or SIP digest authentication with TLS encryption.

- Not supported, as these headers are always caught by the P-CSCF (and Bono doesn't support AKA).

SMS over IP (RFC 3862 and RFC 5438)

- Covers CPIM message format used between UEs and AS implementing SMS-GW function (RFC 3862) and Message Disposition Notifications sent from the SMS-GW to the UE (RFC 5438).
- Transported as body in MESSAGE method across IMS core, so transparent to proxy components. Therefore should be supported by Clearwater once we have tested support for MESSAGE methods.

SIP over SCTP (RFC 4168)

- Defines how SIP can be transported using SCTP (instead of UDP or TCP).
- IMS allows SCTP transport within the core of the network, but not between P-CSCF and UEs.
- Clearwater does not support SCTP transport (nor does PJSIP). Strictly speaking this is relevant to Clearwater, but it's not clear whether anyone would use it.

Signalling pre-emption events (RFC 4411)

- Defines use of the SIP Reason header in BYE messages to signal when a session is being terminated because of a network pre-emption event, for example, if the resources originally acquired for the call were need for a higher priority session.
- Optional according to TS 24.229.
- Not currently supported by Clearwater.

Resource Priority (RFC 4412)

- Covers Resource-Priority and Accept-Resource-Priority headers. Intended to allow UEs to signal high priority calls that get preferential treatment by the network (for example, emergency service use).
- Not currently supported by Clearwater.
- Optional according to TS 24.229.

Service URNs (RFC 5031)

- Defines a URN namespace to identify services. IMS allows UEs to use such service URNs as target URIs when establishing a call. In particular, it is mandatory for UEs to signal emergency calls using a service URN of the form urn:service:sos possibly with a subtype, and the P-CSCF must be able to handle these requests appropriately, routing to an E-CSCF.
- Clearwater currently has no support for service URNs.

Rejecting anonymous requests (RFC 5079)

- Defines a status code used to reject anonymous requests if required by local policy/configuration.
- Optional according to TS 24.229.
- Not currently supported by Clearwater.

Subscribing to events on multiple resources (RFC 5367)

- Allows a SIP node to subscribe to events on multiple resources with a single SUBSCRIBE message by specifying a resource list in the body of the message.
- Optional for any IMS node that can be the target of a SUBSCRIBE request.
- Not currently supported by Clearwater.

Max-Breadth header (RFC 5393)

- Intended to plug an amplification vulnerability in SIP forking. Any forking proxy must limit the breadth of forking to breadth specified in this header.
- According to TS 24.229 is mandatory on any node that supports forking.
- Not currently supported by Clearwater.

Media feature tag for MIME application subtypes (RFC 5688)

- Defines a media feature tag to be used with the mechanisms in RFC 3840 and RFC 3841 to direct requests to UAs that support a specific MIME application subtype media stream.
- According to TS 24.229 support for this feature tag is mandatory on S-CSCFs.
- Not currently supported, but support may drop out of implementing RFC 3840/RFC 3841 (depending on the what match criteria the tag requires).

XCAPdiff event package (RFC 5875)

- Defines an event package allowing applications to get notifications of changes to XCAP documents.
- Used by IMS at Ut reference point to allow UEs to control service settings. According to TS 24.229, mandatory for XDMS server, but optional for UEs.
- Not supported by Homer.

Correct transaction handling of 2xx responses to INVITE (RFC 6026)

- A fix to basic SIP transaction model to avoid INVITE retransmissions being incorrectly identified as a new transaction and to plug a security hole around the forwarding of uncorrelated responses through proxies. The change basically adds a new state to the transaction state machine when previously the transaction would have been considered terminated and therefore deleted.
- This fix is mandatory according to TS 24.229.
- Not supported by Clearwater, and will probably require PJSIP changes.

P-Asserted-Service and P-Preferred-Service headers (RFC 6050)

- Defines a mechanism to allow a UE to signal the service it would like (although this is not binding on the network) and other components to signal between themselves the service being provided. The UE may optionally include a P-Preferred-Service header on any request specifying the service it wishes to use. The S-CSCF is responsible for checking that the service requested in P-Preferred-Service is (a) supported for the subscriber and (b) consistent with the actual request. If the request is allowed, the S-CSCF replaces the P-Preferred-Service

with a P-Asserted-Service header. If either check fails, the S-CSCF may reject the request or it may allow it but ignore the P-Preferred-Service header. If the UE does not specify a P-Preferred-Service header (or the specified one was not valid) the S-CSCF should work out the requested service by analysing the request parameters, and add a P-Asserted-Service header encoding the result.

- In IMS networks, header should contain an IMS Communication Service Identifier (ICSI) - defined values are documented at http://www.3gpp.com/Uniform-Resource-Name-URN-list.html - examples include MMTEL (3gpp-service.ims.icsi.mmtel), IPTV (3gpp-service.ims.icsi.iptv), Remote Access (3gpp-service.ims.icsi.ra), and Converged IP Messaging (3gpp-service.ims.icsi.oma.cpm.* depending on exact service being requested/provided).
- Mandatory function according to TS 24.229.
- Not currently supported by Clearwater.

SIP INFO messages (RFC 6086)

- Framework for exchanging application specification information within a SIP dialog context.
- Not currently supported by Clearwater.
- Optional according to TS 24.229.

Indication of support for keepalives (RFC 6223)

- Adds a parameter to Via headers to allow nodes to agree the type of keepalives to be used to keep NAT pinholes open.
- Mandatory for UEs and P-CSCFs, optional elsewhere.
- Not currently supported by Clearwater and would require PJSIP changes.

Response code for indication of terminated dialog (RFC 6228)

- Defines a new status code to indicate the termination of an early dialog (ie. a dialog created by a provisional response) prior to sending a final response.
- According to TS 24.229 this parameter is mandatory for all UA components than can send or receive INVITEs, and mandatory for S-CSCF because it has implications on forking proxies.
- This is not currently supported by Clearwater.

P-Private-Network-Indication (referenced as draft-vanelburg-sipping-private-network-indication-02)

- Mechanisms for transport of private network information across an IMS core.
- Optional according to TS 24.229.
- Not currently supported by Clearwater.

Session-ID header (referenced as draft-kaplan-dispatch-session-id-00)

- Defines an end-to-end globally unique session identifier that is preserved even for sessions that traverse B2BUAs).
- Optional according to TS 24.229.

- Not currently supported by Clearwater.

Interworking with ISDN (draft-ietf-cuss-sip-uui-06 and draft-ietf-cuss-sip-uui-isdn-04)

- Defines mechanisms/encodings for interworking ISDN with SIP.
- Optional according to TS 24.229.
- Not currently supported by Clearwater.

SDP/Media RFCs

TS 24.229 references a number of specifications which relate to media function - either SDP negotiation or media transport or both. Clearwater currently passes SDP transparently and does not get involved in media flows. Unless we change this position, Clearwater can either be considered to support the RFC (because it supports passing SDP with the relevant contents) or that the RFC is not applicable to Clearwater.

The following is a brief explanation of each RFC, and its relevance to IMS.

- SDP (RFC 4566) - defines basic SDP protocol.
- Offer/Answer model for SDP media negotiation (RFC 3264) - defines how SDP is used to negotiate media.
- Default RTP/AV profile (RFC 3551) - defines default mappings from audio and video encodings to RTP payload types.
- Media Resource Reservation (RFC 3312 and RFC 4032) - defines additional SDP parameters to signal media resource reservation between two SIP UAs. TS 24.229 requires UEs, MGCFs and ASs to use these mechanisms, and that P-CSCF should monitor the flows if it is performing IMS-ALG or media gating functions.
- Mapping of media streams to resource reservation (RFC 3524 and RFC 3388) - define how multiple media streams can be grouped in SDP and mapped to a single resource reservation. IMS requires UEs to use these encodings when doing resource reservations.
- Signaling bandwidth required for RTCP (RFC 3556) - by default RTCP is assumed to consume 5% of session bandwidth, but this is not accurate for some encodings, so this RFC defines explicit signaling of RTCP bandwidth in SDP. This function is optional according to TS 24.229.
- TCP media transport (RFC 3556 and RFC 6544) - defines how support for TCP media transport is encoded in SDP for basic SDP exchanges and for ICE exchanges. According to TS 24.229, media over TCP is optional in most of an IMS network, but mandatory in an IBCF implementing IMS-ALG function.
- ICE (RFC 5245) - defines ICE media negotiation used to establish efficient media paths through NATs. According to TS 24.229 support for ICE is optional in most of the IMS network, but mandatory on an IBCF implementing IMS-ALG function.
- STUN (RFC 5389) - defines a protocol used to allow UAs to obtain their server reflexive address for use in ICE.
- TURN (RFC 5766) - defines extensions to STUN used to relay media sessions via a TURN server to traverse NATs when STUN-only techniques don't work.
- Indicating support for ICE in SIP (RFC 5768) - defines a media feature tag used to signal support for ICE.
- SDP for binary floor control protocol (RFC 4583) - defines SDP extensions for establishing conferencing binary floor control protocol streams. Optional according to TS 24.229.
- Real-time RTCP feedback (RFC 4585 and RFC 5104) - defines extensions to RTCP to provide extended real-time feedback on network conditions. Mandatory for most IMS components handling media (MRFs, IBCFs, MGCFs), but optional for UEs.

- SDP capability negotiation (RFC 5939) - defines extensions to SDP to allow for signaling and negotiation of media capabilities (such as alternate transports - SRTP). Mandatory for most IMS components handling media (MRFs, IBCFs, MGCFs), but optional for UEs.
- SDP transport independent bandwidth signaling (RFC 3890) - defines extensions to SDP to signal codec-only (ie. independent of transport) bandwidth requirements for a stream. Optional for IMS components handling media.
- Secure RTP (RFC 3711, RFC 4567, RFC 4568, RFC 6043) - defines transport of media over a secure variant of RTP, supporting encryption (to prevent eavesdropping) and integrity protecting (to protect against tampering). Keys are either exchanged in SDP negotiation (specified in RFC 4567 and RFC 4568) or distributed via a separate key management service - termed MIKEY-TICKEY (specified in RFC 6043).
- Transcoder invocation using 3PCC (RFC 4117) - defines how a transcoding service can be inserted into the media path when required using third-party call control flows. According to TS 24.229 this is only applicable to app servers and MRFC, and even then is optional.
- MSRP (RFC 4975) - defines a protocol for session-based transmission of a sequence of instant messages. Only applicable to UEs, ASs and MRFCs and optional.
- SDP for file transfer (RFC 5547) - defines SDP extensions to support simple file transfers between two IMS nodes. Optional.
- Explicit congestion notifications for RTP over UDP (RFC 6679) - defines RTCP extensions for reporting urgent congestion conditions and reporting congestion summaries. Optional.
- Setting up audio streams over circuit switched bearers (referenced as draft-ietf-mmusic-sdp-cs-00) - defines SDP extensions for negotiating audio streams over a circuit switched network. Mandatory for ICS capable UEs and SCC-AS, otherwise optional.
- Media capabilities negotiation in SDP (referenced as draft-ietf-mmusic-sdp-media-capabilities-08) - defines SDP extensions for signaling media capabilities such as encodings and formats. Mandatory for ICS capable UEs and SCC-AS, otherwise optional.
- Miscellaneous capabilities negotiation in SDP (referenced as draft-ietf-mmusic-sdp-miscellaneous-caps-00) - defines SDP extensions for signaling some miscellaneous capabilities. Mandatory for ICS capable UEs and SCC-AS, otherwise optional.

Not Relevant to Clearwater

Locating P-CSCF using DHCP (RFC 3319 and RFC 3361)

- These RFCs describe a mechanism for locating a SIP proxy server using DHCP. (RFC 3319 defines the mechanism for IPv6/DHCPv6, and RFC 3361 is the IPv4 equivalent - not sure why they happened that way round!).
- The IMS specifications allows this as one option for a UE to locate a P-CSCF, although there are other options such as manual configuration of a domain name or obtaining it from some access-network specific mechanism.
- This is irrelevant to Clearwater - there would be no point in Clearwater providing a DHCP server with this function as there will be existing mechanisms used by clients to obtain their own IP addresses. A service provider might enhance their own DHCP servers to support this function if required.

Proxy-to-proxy SIP extensions for PacketCable DCS (RFC 3603)

- Only relevance to IMS is that it defines a billing correlation parameter (bcid) which is passed in the P-Charging-Vector header from DOCSIS access networks.

Geolocation (RFC 4119 and RFC 6442)

- Frameworks for passing geo-location information within SIP messages - RFC 4119 encodes geo-location in a message body, RFC 6442 encodes a URI reference where the UEs location can be found.
- According to TS 24.229 only required on an E-CSCF.

P-User-Database header (RFC 4457)

- Used when an IMS core has multiple HSSs and an SLF - allows I-CSCF to signal to S-CSCF which HSS to use to avoid multiple SLF look-ups.
- Optional according to TS 24.229.
- Not applicable to Clearwater given its stateless architecture.

URIs for Voicemail and IVR applications (RFC 4458)

- Defines conventions for service URIs for redirection services such as voicemail and IVR.
- Optional according to TS 24.229.

Emergency call requirements and terminology (RFC 5012)

- Referenced to define some terminology used in IMS architecture for handling emergency calls.

Answering Modes (RFC 5373)

- Defines a mechanism for a caller to control the answer mode at the target of the call. Use cases can include invoking some kind of auto-answer loopback. Covers the Answer-Mode and Priv-Answer-Mode headers.
- In general is transparent to proxies (provided proxies pass headers through), but the RFC recommends the mechanism is not used in environments that support parallel forking.
- Optional according to TS 24.229 - and arguably not a good idea because of bad interactions with forking.

Clearwater Automatic Clustering and Configuration Sharing

Clearwater has a feature that allows nodes in a deployment to automatically form the correct clusters and share configuration with each other. This makes deployments much easier to manage. For example:

- It is easy to add new nodes to an existing deployment. The new nodes will automatically join the correct clusters according to their node type, without any loss of service. The nodes will also learn the majority of their config from the nodes already in the deployment.
- Similarly, removing nodes from a deployment is straightforward. The leaving nodes will leave their clusters without impacting service.
- It makes it much easier to modify configuration that is shared across all nodes in the deployment.

This features uses etcd as a decentralized data store, a `clearwater-cluster-manager` service to handle automatic clustering, and a `clearwater-config-manager` to handle configuration sharing.

Etcd masters and proxies

Clearwater nodes can run either as an etcd master or an etcd proxy. When deploying a node, you can chose whether it acts as a master or proxy by filling in either the `etcd_cluster` or `etcd_proxy` config option in `/etc/clearwater/local_config` (see the configuration options reference for more details).

There are some restrictions on which nodes can be masters or proxies:

- There must always be at least 3 etcd masters in the cluster
- The first node to be deployed in a site must be an etcd master

The automated and manual install instructions will both create a deployment with all nodes running as etcd masters.

Provisioning Subscribers

Clearwater provides the Ellis web UI for easy provisioning of subscribers. However, sometimes a more programmatic interface is desirable.

Homestead-Prov provides a provisioning API but, for convenience, Clearwater also provides some command-line provisioning tools.

By default, the tools are installed on the Dime nodes only (as part of the clearwater-prov-tools package), in the `/usr/share/clearwater/bin` directory.

There are 5 tools.

- `cw-create_user` - for creating users
- `cw-update_user` - for updating users' passwords
- `cw-delete_user` - for deleting users
- `cw-display_user` - for displaying users' details
- `cw-list_users` - for listing users

Creating users

New users can be created with the `cw-create_user` tool. As well as creating single users, it's also possible to create multiple users with a single command. Note that this is not recommended for provisioning large numbers of users - for that, bulk provisioning is much quicker.

```
usage: create_user.py [-h] [-k] [--hsprov IP:PORT] [--plaintext]
                      [--ifc iFC-FILE] [--prefix TWIN_PREFIX]
                      <directory-number>[..<directory-number>] <domain>
                      <password>

Create user

positional arguments:
  <directory-number>[..<directory-number>]
  <domain>
  <password>

optional arguments:
  -h, --help            show this help message and exit
  -k, --keep-going      keep going on errors
  --hsprov IP:PORT      IP address and port of homestead-prov
```

```
  --plaintext           store password in plaintext
  --ifc iFC-FILE        XML file containing the iFC
  --prefix TWIN_PREFIX  twin-prefix (default: 123)
```

Update users

Existing users' passwords can be updated with the `cw-update_user` tool.

```
usage: update_user.py [-h] [-k] [-q] [--hsprov IP:PORT] [--plaintext]
                      <directory-number>[..<directory-number>] <domain>
                      <password>

Update user

positional arguments:
  <directory-number>[..<directory-number>]
  <domain>
  <password>

optional arguments:
  -h, --help            show this help message and exit
  -k, --keep-going      keep going on errors
  -q, --quiet           don't display the user
  --hsprov IP:PORT      IP address and port of homestead-prov
  --plaintext           store password in plaintext
```

Delete users

Users can be deleted with the `cw-delete_user` tool.

```
usage: delete_user.py [-h] [-f] [-q] [--hsprov IP:PORT]
                      <directory-number>[..<directory-number>] <domain>

Delete user

positional arguments:
  <directory-number>[..<directory-number>]
  <domain>

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           proceed with delete in the face of errors
  -q, --quiet           silence 'forced' error messages
  --hsprov IP:PORT      IP address and port of homestead-prov
```

Display users

Users' details can be displayed with the `cw-display_user` tool.

```
usage: display_user.py [-h] [-k] [-q] [-s] [--hsprov IP:PORT]
                       <directory-number>[..<directory-number>] <domain>

Display user
```

```
positional arguments:
  <directory-number>[..<directory-number>]
  <domain>

optional arguments:
  -h, --help            show this help message and exit
  -k, --keep-going      keep going on errors
  -q, --quiet           suppress errors when ignoring them
  -s, --short           less verbose display
  --hsprov IP:PORT      IP address and port of homestead-prov
```

List users

All the users provisioned on the system can be listed with the `cw-list_users` tool.

Note that the `--full` parameter defaults to off because it greatly decreases the performance of the tool (by more than an order of magnitude).

The `--pace` parameter's default values should ensure that this does not use more than 10% of the Dime cluster's CPU - that is 5 users per second if `--force` is set and 500 if not. If you set the "–pace" parameter to more than the default, you'll be prompted to confirm (or specify the `--force` parameter).

```
usage: list_users.py [-h] [-k] [--hsprov IP:PORT] [--full] [--pace PACE] [-f]

List users

optional arguments:
  -h, --help        show this help message and exit
  -k, --keep-going  keep going on errors
  --hsprov IP:PORT  IP address and port of homestead-prov
  --full            displays full information for each user
  --pace PACE       sets the target number of users to list per second
  -f, --force       forces specified pace
```

Dealing with Failed Nodes

Nodes can be easily removed from a Clearwater deployment by following the instructions for elastic scaling. However, sometimes a node or nodes may fail unexpectedly. If the nodes cannot be recovered, then you should do the following (in the order specified):

- If one or more nodes have failed that were acting as etcd masters (see configuration) and as a result you have lost 50% (or more) of your etcd master nodes in any one site then the etcd cluster for that site will have lost "quorum" and have become read-only. To recover the etcd cluster you will need to follow the process here.
- If one or more nodes have failed that were acting as etcd masters but *more* than half of your etcd cluster remains operational then you must first follow the steps below: "Removing a failed node from an etcd cluster"
- If a Vellum node has failed then you should follow the instructions below: "Removing a failed Vellum node from the data store clusters"
- You can now spin up a new node to replace the lost capacity. If you are replacing a node that had been acting as an etcd master then you should typically configure the new node to also be an etcd master in order to retain your original etcd cluster size.

The processes described below do not affect call processing and can be run on a system handling call traffic.

Removing a failed node from an etcd cluster

If a node fails that was acting as an etcd master then it must be manually removed from the site's etcd cluster. Failure to do so may leave the site in a state where future scaling operations do not work, or where in-progress scaling operations fail to complete.

This process assumes that more than half of the site's etcd cluster is still healthy and so the etcd cluster still has quorum. If 50% or more of the etcd masters in a given site have failed then you will need to first follow the process here.

For each failed node, log into a healthy etcd master in the same site as the failed node and run the following steps:

- Run `clearwater-etcdctl cluster-health` and make a note of the ID of the failed node.
- Run `clearwater-etcdctl member list` to check that the failed node reported is the one you were expecting (by looking at its IP address).
- Run `clearwater-etcdctl member remove <ID>`, replacing `<ID>` with the ID learned above.

Removing a failed Vellum node from the data store clusters

If one or more Vellum nodes fail then they will need to be removed from all of the data store clusters that they were part of. You will need to know the IP addresses of the failed nodes and if you are using separate signaling and management networks, you must use the signaling IP addresses.

For each site that contains one or more failed Vellum nodes, log into a healthy Vellum node in the site and run the following commands. If the site contains multiple failed nodes then you will need to run each command for all failed nodes in the site before moving on to the next command, and each command will block until you have run it for all of the failed nodes in a site so you will need to open a separate shell session for each node that has failed.

- `sudo cw-mark_node_failed "vellum" "memcached" <failed node IP>`
- `sudo cw-mark_node_failed "vellum" "chronos" <failed node IP>`

If you are using any of Homestead-Prov, Homer or Memento, also run: * `sudo cw-mark_node_failed "vellum" "cassandra" <failed node IP>`

Dealing with Multiple Failed Nodes

If any site in your deployment loses half or more of its etcd master nodes permanently, it loses "quorum". This means that the underlying etcd cluster becomes read-only. While the etcd cluster is in this state, you can't perform any scaling operations or change configuration and have it synced across the deployment. You should use this process to recover the etcd cluster in the failed site.

If you haven't lost half (or more) of your etcd master nodes in a site, then you can use the process described here for each of your failed nodes.

Procedure

The procedure creates a new etcd cluster to replace the existing cluster. The new cluster is populated with the configuration saved in the configuration files on disk. This allows us to recreate the cluster, even in cases when the existing cluster is too badly corrupted to read from.

This procedure won't impact service. You should follow this process completely - the behaviour is unspecified if this process is started but not completed. It is always safe to restart this process from the beginning (for example, if you encounter an error partway through).

If there are no live Vellum nodes in the site, you should continue with this process, missing out the steps that require running commands on Vellum. Once the etcd cluster is recovered, you should add the new replacement nodes.

Stop the etcd processes

Stop the etcd processes on every node in the affected site.

- Run `sudo monit stop -g etcd`
- Run `sudo monit stop -g clearwater_cluster_manager`
- Run `sudo monit stop -g clearwater_config_manager`
- Run `sudo monit stop -g clearwater_queue_manager`
- Run `sudo touch /etc/clearwater/no_cluster_manager`
- Run `sudo rm -rf /var/lib/clearwater-etcd/*`

Select your master nodes

To follow this process you need to choose some nodes to be the new masters of the etcd cluster:

- If you have 3 or more working Vellum nodes in the site, you should use those
- If not, you should use all the nodes in the site

Check the configuration on your nodes

The next step is to ensure that the configuration files on each node are correct.

Any of the master nodes - Shared configuration

The shared configuration is at `/etc/clearwater/shared_config`. Verify that this is correct, then copy this file onto every other master node. Please see the configuration options reference for more details on how to set the configuration values.

Vellum - Chronos configuration

- The configuration file is at `/etc/chronos/chronos_cluster.conf`.
- Verify that this is present and syntactically correct on all Vellum nodes in the affected site.
  - This should follow the format here.
  - If the file isn't present, or is invalid, then make the configuration file contain all Vellum nodes in the site as nodes.
  - Otherwise, don't change the states of any nodes in the file (even if you know the node has failed).
- If there is more than one failed node then there will be timer failures until this process has been completed. This could prevent subscribers from receiving notifications when their registrations/subscriptions expire.

Vellum - Memcached configuration

- The configuration file is at `/etc/clearwater/cluster_settings`.
- Verify that this is present and syntactically correct on all Vellum nodes in the affected site.
  - This can have a `servers` line and a `new_servers` line - each line has the format `<servers|new_servers>=<ip address>,<ip address>, ...`
  - If the file isn't present, or is invalid, then make the configuration file contain all Vellum nodes in the site on the `servers` lines, and don't add a `new_servers` line.
  - Otherwise, don't change the states of any nodes in the file (even if you know the node has failed).
- If there is more than one failed node (and there is no remote site, or more than one failed node in the remote site) then there will be registration and call failures, and calls will be incorrectly billed (if using Ralf) until this process has been completed.

Vellum - Cassandra configuration

If you are using any of Homestead-Prov, Homer or Memento, check that the Cassandra cluster is healthy by running the following on a Vellum node:

```
sudo /usr/share/clearwater/bin/run-in-signaling-namespace nodetool status
```

If the Cassandra cluster isn't healthy, you must fix this up before continuing, and remove any failed nodes.

Sprout - JSON configuration

Check the JSON configuration files on all Sprout nodes in the affected site:

- Verify that the `/etc/clearwater/enum.json` file is correct, fixing it up if it's not.
- Verify that the `/etc/clearwater/s-cscf.json` file is correct, fixing it up if it's not.
- Verify that the `/etc/clearwater/bgcf.json` file is correct, fixing it up if it's not.

Sprout - XML configuration

Check the XML configuration files on all Sprout nodes in the affected site:

- Verify that the `/etc/clearwater/shared_ifcs.xml` file is correct, fixing it up if it's not.
- Verify that the `/etc/clearwater/fallback_ifcs.xml` file is correct, fixing it up if it's not.

Running one of the commands `sudo cw-validate_{shared|fallback}_ifcs_xml` will check if the specified file is syntactically correct.

Recreate the etcd cluster

- On your selected master nodes, set `etcd_cluster` in `/etc/clearwater/local_config` to a comma separated list of the management IP addresses of your master nodes.
- Start etcd on the master nodes
  - Run `sudo monit monitor -g etcd`
  - Run `sudo monit monitor -g clearwater_config_manager`

  - Run `sudo monit monitor -g clearwater_queue_manager`
- This creates the etcd cluster, and synchronises the shared configuration. It doesn't recreate the data store cluster information in etcd yet.
- Verify that the master nodes have formed a new etcd cluster successfully:
  - Running `sudo monit summary` on each master node should show that the etcd processes are running successfully, except the `clearwater_cluster_manager_process`
  - Running `sudo clearwater-etcdctl cluster-health` (on a single master node) should show that the etcd cluster is healthy
  - Running `sudo clearwater-etcdctl member list` should show that all the master nodes are members of the etcd cluster.
- Verify that the configuration has successfully synchronized by running `sudo /usr/share/clearwater/clearwater-config-manager/scripts/check_config_sync`

Add the rest of the nodes to the etcd cluster

Run this process on every node which is not one of the master nodes in the affected site in turn. If all nodes in the site are master nodes, you can skip this step.

- Set `etcd_proxy` in `/etc/clearwater/local_config` to a comma separated list of the management IP addresses of your master nodes.
- Start etcd on the node
  - Run `sudo monit monitor -g etcd`
  - Run `sudo monit monitor -g clearwater_config_manager`
  - Run `sudo monit monitor -g clearwater_queue_manager`
- Verify that the node has contacted the etcd cluster successfully:
  - Running `sudo monit summary` should show that the etcd processes are running successfully, except the `clearwater_cluster_manager_process`

Recreate the data store cluster values in etcd

Run these commands on one Vellum node in the affected site:

```
/usr/share/clearwater/clearwater-cluster-manager/scripts/load_from_chronos_cluster␣
↪vellum
/usr/share/clearwater/clearwater-cluster-manager/scripts/load_from_memcached_cluster␣
↪vellum
```

If you are using any of Homestead-Prov, Homer or Memento, also run:

```
/usr/share/clearwater/clearwater-cluster-manager/scripts/load_from_cassandra_cluster␣
↪vellum
```

Verify the cluster state is correct in etcd by running `sudo /usr/share/clearwater/clearwater-cluster-manager/scripts/check_cluster_state`

Start the cluster manager on all nodes

Run this process on every node (including the master nodes) in the affected site in turn.

- Run `sudo rm /etc/clearwater/no_cluster_manager`
- Run `sudo monit monitor -g clearwater_cluster_manager`
- Verify that the cluster-manager comes back up by running `sudo monit summary`.

Next steps

Your deployment now has a working etcd cluster. You now need to:

- Remove the failed nodes from the data store clusters for Chronos and Memcached (following http://clearwater.readthedocs.io/en/latest/Handling_Failed_Nodes.html#removing-a-node-from-a-data-store).
- Recover redundancy by replacing the failed nodes.

Dealing with Complete Site Failure

In a geographically redundant deployment, you may encounter the situation where an entire site has permanently failed (e.g. because the location of that geographic site has been physically destroyed). This article covers:

- How to recover from this situation
- The likely impact of a site failure

More information about Clearwater's geographic redundancy support is available here and information on how to configure a geographically redundant deployment is available here.

Recovery

If you are using any of Homestead-Prov, Homer or Memento, to recover from this situation all you need to do is remove the failed Vellum nodes from Cassandra cluster.

```
* From any Vellum node in the remaining site, run `cw-remove_site_from_cassandra
↪<site ID - the name of the failed site>`
```

If you are not using any of Homestead-Prov, Homer or Memento, you do not need to do anything to recover the single remaining site.

You should now have a working single-site cluster, which can continue to run as a single site, or be safely paired with a new remote site (details on how to set up a new remote site are here).

Impact

This section considers the probable impact on a subscriber of a total outage of a region in a 2-region geographically-redundant deployment. It assumes that the deployments in both regions have sufficient capacity to cope with all subscribers (or the deployments scale elastically).

The subscriber interacts with Clearwater through 3 interfaces, and these each have a different user experience.

- SIP to Bono for calls
- HTTP to Homer for direct call service configuration

- HTTP to Ellis for web-UI-based provisioning

For the purposes of the following descriptions, we label the two regions A and B, and the deployment in region A has failed.

SIP to Bono

If the subscriber is connected to a Bono node in region A, their TCP connection fails. The behavior is client-specific, but we assume that the client will attempt to re-register with the region A Bono, fail to connect, and attempt connections to the other Bonos returned by DNS domain query until they locate a Bono node in region B, at which point their re-register succeeds and service is restored.

If the subscriber is connected to a Bono node in region B, their TCP connection does not fail, they do not need to re-register and their service is unaffected.

Realistically however, if 50% of subscribers all re-registered almost simultaneously (due to their TCP connection dropping and their DNS being timed out), it's unlikely that Bono would be able to keep up.

HTTP to Homer

If the subscriber was using a Homer node in region A, their requests would fail until their DNS timed out, and they retried to a Homer in region B. If the subscriber was using a Homer node in region B, they would see no failures.

HTTP to Ellis

Ellis is not geographically redundant. If Ellis was deployed in region A, all subscriber provisioning would fail until region A was recovered. If Ellis was deployed in region B, there would be no outage.

C++ Coding Guidelines

Naming

- Class names are CamelCase, beginning with a capital letter.
- Constant names are ALL_CAPITALS with underscores separating words.
- Method, method parameter and local variable names are lowercase with underscores separating words, e.g. `method_name`.
- Member variable names are lowercase with underscores separating words and also begin with an underscore, e.g. `_member_variable`.

C++ Feature Use

- We assume a C++11-compliant compiler, so C++11 features are allowed with some exceptions.
- No use of `auto` type declarations.
- No use of `using namespace ...`, if the namespace is particularly lengthy, consider using namespace aliasing (e.g. `namespace po = boost::program_options`).
- Avoid using Boost (or similar) libraries that return special library-specific pointers, to minimize "infection" of the code-base. Consider using the C++11 equivalents instead.

Formatting

C++ code contributed to Clearwater should be formatted according to the following conventions:

- Braces on a separate line from function definitions, `if` statements, etc.
- Two-space indentation
- Pointer operators attached to the variable type (i.e. `int* foo` rather than `int *foo`)
- `if` blocks must be surrounded by braces

For example:

```
if (x)
      int *foo = do_something();
```

will be replaced with

```
if (x)
{
 int* foo = do_something();
}
```

It's possible to fix up some code automatically using astyle, with the options `astyle --style=ansi -s2 -M80 -O -G -k1 -j -o`. This fixes up a lot of the most common errors (brace style, indentation, overly long lines), but isn't perfect - there are some cases where breaking the rules makes the code clearer, and some edge cases (e.g. around switch statements and casts on multiple lines) where our style doesn't always match astyle's.

Language Features

- Use of the `auto` keyword is forbidden.

Commenting

Where it is necessary to document the interface of classes, this should be done with Doxygen-style comments - three slashes and appropriate `@param` and `@returns` tags.

```
/// Apply first AS (if any) to initial request.
//
// See 3GPP TS 23.218, especially s5.2 and s6, for an overview of how
// this works, and 3GPP TS 24.229 s5.4.3.2 and s5.4.3.3 for
// step-by-step details.
//
// @Returns whether processing should stop, continue, or skip to the end.
AsChainLink::Disposition
AsChainLink::on_initial_request(CallServices* call_services,
```

Ruby Coding Guidelines

Strongly based on https://github.com/chneukirchen/styleguide/ with some local changes.

Formatting

- Use UTF-8 encoding in your source files.
- Use 2 space indent, no tabs.
- Use Unix-style line endings, including on the last line of the file.
- Use spaces around operators, after commas, colons and semicolons, around { and before }.
- No spaces after (, [and before],).
- Prefer postfix modifiers (if, unless, rescue) when possible.
- Indent when as deep as case then indent the contents one step more.
- Use an empty line before the return value of a method (unless it only has one line), and an empty line between defs.
- Use Yard and its conventions for API documentation. Don't put an empty line between the comment block and the definition.
- Use empty lines to break up a long method into logical paragraphs.
- Keep lines shorter than 80 characters.
- Avoid trailing whitespace.

Syntax

- Use def with parentheses when there are arguments.
- Conversely, avoid parentheses when there are none.
- Never use for, unless you exactly know why. Prefer each or loop.
- Never use then, a newline is sufficient.
- Prefer words to symbols.
- and and or in place of && and ||
- not in place of !
- Avoid ?:, use if (remember: if returns a value, use it).
- Avoid if not, use unless.
- Suppress superfluous parentheses when calling methods, unless the method has side-effects.
- Prefer do...end over {...} for multi-line blocks.
- Prefer {...} over do...end for single-line blocks.
- Avoid chaining function calls over multiple lines (implying, use {...} for chained functions.
- Avoid return where not required.
- Avoid line continuation (\) where not required.
- Using the return value of = is okay.
- if v = array.grep(/foo/)
- Use ||= freely for memoization.
- When using regexps, freely use =~, -9, :math:‘~, ‘ and $‘ when needed.

- Prefer symbols (:name) to strings where applicable.

Naming

- Use snake_case for methods.
- Use CamelCase for classes and modules. (Keep acronyms like HTTP, RFC and XML uppercase.)
- Use SCREAMING_CASE for other constants.
- Use one-letter variables for short block/method parameters, according to this scheme:
- a,b,c: any object
- d: directory names
- e: elements of an Enumerable or a rescued Exception
- f: files and file names
- i,j: indexes or integers
- k: the key part of a hash entry
- m: methods
- o: any object
- r: return values of short methods
- s: strings
- v: any value
- v: the value part of a hash entry
- And in general, the first letter of the class name if all objects are of that type (e.g. `nodes.each { |n| n.name }`)
- Use _ for unused variables.
- When defining binary operators, name the argument other.
- Use def self.method to define singleton methods.

Comments

- Comments longer than a word are capitalized and use punctuation. Use two spaces after periods.
- Avoid superfluous comments. It should be easy to write self-documenting code.

Code design

- Avoid needless meta-programming.
- Avoid long methods. Much prefer to go too far the wrong way and have multiple one-line methods.
- Avoid long parameter lists, consider using a hash with documented defaults instead.
- Prefer functional methods over procedural ones (common methods below):
- each - Apply block to each element
- map - Apply block to each element and remember the returned values.

- select - Find all matching elements
- detect - Find first matching element
- inject - Equivalent to foldl from Haskell
- Use the mutating version of functional methods (e.g. map!) where applicable, rather than using temporary variables.
- Avoid non-obvious function overloading (e.g. don't use ["0"] * 8 to initialize an array).
- Prefer objects to vanilla arrays/hashes, this allows you to document the structure and interface.
- Protect the internal data stores from external access. Write API functions explicitly.
- Use attr_accessor to create getters/setters for simple access.
- Prefer to add a to_s function to an object for ease of debugging.
- Internally, use standard libraries where applicable (See the docs for the various APIs).:
- Hash, Array and Set
- String
- Fixnum and Integer
- Thread and Mutex
- Fiber
- Complex
- Float
- Dir and File
- Random
- Time
- Prefer string interpolation "blah#{expr}" rather than appending to strings.
- Prefer using the %w{} family of array generators to typing out arrays of strings manually.

General

- Write ruby -w safe code.
- Avoid alias, use alias_method if you absolutely must alias something (for Monkey Patching).
- Use OptionParser for parsing command line options.
- Target Ruby 2.0 (except where libraries are not compatible, such as Chef).
- Do not mutate arguments unless that is the purpose of the method.
- Do not mess around in core classes when writing libraries.
- Do not program defensively.
- Keep the code simple.
- Be consistent.
- Use common sense.

Debugging Clearwater with GDB and Valgrind

Valgrind

Valgrind is a very powerful profiling and debugging tool.

Before you run Bono or Sprout under valgrind, you may want to tweak pjsip's code slightly as indicated below, then rebuild and patch your nodes.

- Valgrind's memory access tracking hooks into malloc and free. Unfortunately, pjsip uses its own memory management functions, and so mallocs/frees relatively rarely. To disable this, modify `pjlib/src/pj/pool_caching.c`'s `pj_caching_pool_init` function to always set cp->max_capacity to 0.
- Valgrind's thread safety tool (helgrind) tracks the order in which locks are taken, and reports on any lock cycles (which can in theory cause deadlocks). One of these locks generates a lot of benign results. To prevent these edit `pjsip\include\pjsip\sip_config.h` and set the value of `PJSIP_SAFE_MODULE` to 0.
- Valgrind's thread safety tool also spots variables that are accessed on mutliple threads without locking the locking necessary to prevent race conditions. Pjsip's memory pools maintain some statistics that are not used by Clearwater, but that trip valgrind's race condition detection. To suppress this edit `pjlib/src/pj/pool_caching.c` and remove the bodies of the `cpool_on_block_alloc` and `cpool_on_block_free` (keeping only the bodies).

To run Bono, Sprout or Homestead under valgrind (the example commands assume you are running sprout), the easiest way is to:

- Find the command line you are using to run Sprout (`ps -eaf | grep sprout`)
- Make sure valgrind is installed on your system and you have the appropriate debug packages installed (`sudo apt-get install valgrind` and `sudo apt-get install sprout-node-dbg`)
- Disable monitoring of sprout (`sudo monit unmonitor -g sprout`)
- Stop sprout (`sudo service sprout stop`)
- Allow child processes to use more file descriptors, and become the sprout user (`sudo -i; ulimit -Hn 1000000; ulimit -Sn 1000000; (sudo -u sprout bash);`)
- Change to the `/etc/clearwater` directory
- Set up the library path (`export LD_LIBRARY_PATH=/usr/share/clearwater/sprout/lib:$LD_LIBRARY_PATH`)
- Run the executable under valgrind, enabling the appropriate valgrind options - for example, to use massif to monitor the Sprout heap `valgrind --tool=massif --massif-out-file=/var/log/sprout/massif.out.%p /usr/share/clearwater/bin/sprout <parameters>` (the –massif-out-file option is required to ensure the output is written to a directory where the sprout user has write permission). If any of Sprout parameters include a semi-colon, you must prefix this with a backslash otherwise the bash interpreter will interpret this as the end of the command.

Valgrind will slow down the running of Bono, Sprout and Homestead by a factor of 5-10. It will produce output when it detects invalid/illegal memory access - often these turn out to be benign, but they're rarely spurious.

GDB

Installing

To install gdb, simply type `sudo apt-get install gdb`. gdb is already installed on build machines, but not on live nodes.

If you're debugging on a live node, it's also worth installing the sprout or bono debug packages. When we build the standard (release) versions, we strip all the symbols out and these are saved off separately in the debug package. Note that you will still be running the release code - the debug symbols will just be loaded by gdb when you start it up. To install these packages, type `sudo apt-get install sprout-node-dbg` or `sudo apt-get install bono-node-dbg`.

Pull Request Process

Dev and master branches

Each component/repository (with a few exceptions) has 2 main branches, `dev` and `master`. Whenever a commit is pushed to the `dev` branch, Jenkins will automatically run the unit tests for the repository and if they pass, merge into `master`.

Development process

Features or any changes to the codebase should be done as follows:

1. Pull the latest code in the `dev` branch and create a feature branch off this.
   - Alternatively, if you want to be sure of working from a good build and don't mind the risk of a harder merge, branch off `master`.
   - If the codebase doesn't have a dev branch, branch off `master`.
2. Implement your feature (including any necessary UTs etc). Commits are cheap in git, try to split up your code into many, it makes reviewing easier as well as for saner merging.
   - If your commit fixes an existing issue #123, include the text "fixes #123" in at least one of your commit messages. This ensures the pull request is attached to the existing issue.
3. Once complete, ensure the following check pass (where relevant):
   - All UTs (including coverage and valgrind) pass.
   - Your code builds cleanly into a Debian package on the repo server.
   - The resulting package installs cleanly using the clearwater-infrastructure script.
   - The live tests pass.
4. Push your feature branch to GitHub.
5. Create a pull request using GitHub, from your branch to `dev` (never `master`, unless the codebase has no dev branch).
6. Await review.
   - Address code review issues on your feature branch.
   - Push your changes to the feature branch on GitHub. This automatically updates the pull request.
   - If necessary, make a top-level comment along the lines "Please re-review", assign back to the reviewer, and repeat the above.
   - If no further review is necessary and you have the necessary privileges, merge the pull request and close the branch. Otherwise, make a top-level comment and assign back to the reviewer as above.

Reviewer process

- Receive notice of review by GitHub email, GitHub notification, or by checking all your assigned GitHub issues.
- Make comments on the pull request (either line comments or top-level comments).
- Make a top-level comment saying something along the lines of "Fine; some minor comments" or "Some issues to address before merging".
- If there are no issues, merge the pull request and close the branch. Otherwise, assign the pull request to the developer and leave this to them.

Unit tests

Project Clearwater has various types of regression testing.

We have live tests which we run regularly over our deployments. We have FV tests that we run any time the underlying code changes. We also have unit tests; these are specific to a repository, and are run anytime any changes are made to the repository.

This document describes how to run the unit tests for the different Clearwater repositories.

C++ Unit Tests

For our components written in C++, we use a common makefile infrastructure.

To run the unit tests, change to the `src` subdirectory below the top-level component directory and issue `make test`.

The unit tests use the Google Test framework, so the output from the test run looks something like this.

```
[==========] Running 92 tests from 20 test cases.
[----------] Global test environment set-up.
[----------] 1 test from AuthenticationTest
[ RUN      ] AuthenticationTest.NoAuthorization
[       OK ] AuthenticationTest.NoAuthorization (27 ms)
[----------] 1 test from AuthenticationTest (27 ms total)

[----------] 6 tests from SimServsTest
[ RUN      ] SimServsTest.EmptyXml
[       OK ] SimServsTest.EmptyXml (1 ms)
...
[ RUN      ] SessionCaseTest.Names
[       OK ] SessionCaseTest.Names (0 ms)
[----------] 1 test from SessionCaseTest (0 ms total)

[----------] Global test environment tear-down
[==========] 92 tests from 20 test cases ran. (27347 ms total)
[  PASSED  ] 92 tests.
```

There are various other options for the unit tests as well:

- Passing `JUSTTEST=testname` just runs the specified test case.
- Passing `NOISY=T` prints the logs made during the test run to the screen. You can set what severity of logs are printed by adding a logging level; the logging level matches the severity levels for our logs (defined here). For example, `NOISY=T:2` prints all logs up to STATUS severity and `NOISY=T:5` prints all logs up to DEBUG severity.

- `make coverage_check` runs code coverage checks (using gcov), and reports if the coverage is less than expected.
- `make coverage_raw` outputs coverage information for each source file.
- `make valgrind` runs memory leak checks (using Valgrind).
- `make full_test` runs the coverage and memory leak checks as well as the tests.

Python Unit Tests

Our components written in Python don't share a common makefile infrastructure. However, each component typically uses the same `make` commands for running the tests. In the top level component directory, run:

- `make test` to run all of the test cases.
- `make coverage` to view the current unit test coverage.
- `make verify` to run flake8 over the code to detect errors.

We do not use tools like pylint by default, as they can be too aggressive and provide a large number of benign errors, often in third party modules. However, it can be difficult to track down import errors using the standard test infrastructure we have in place, and it can be useful under these circumstances to be able to run pylint to draw out any bad imports that might be causing the modules or unit tests to fail.

For example, if we deliberately break an import in the cluster_manager plugin_base.py file (changing `from abc import ...` to `from NOTabc import ...`), running `make test` gives the following output:

```
Traceback (most recent call last):
  File "cluster_mgr_setup.py", line 52, in <module>
    tests_require=["pbr==1.6", "Mock"],
  File "/usr/lib/python2.7/distutils/core.py", line 151, in setup
    dist.run_commands()
  File "/usr/lib/python2.7/distutils/dist.py", line 953, in run_commands
    self.run_command(cmd)
  File "/usr/lib/python2.7/distutils/dist.py", line 972, in run_command
    cmd_obj.run()
  File "build/bdist.linux-x86_64/egg/setuptools/command/test.py", line 170, in run
  File "build/bdist.linux-x86_64/egg/setuptools/command/test.py", line 191, in run_
↪tests
  File "/usr/lib/python2.7/unittest/main.py", line 94, in __init__
    self.parseArgs(argv)
  File "/usr/lib/python2.7/unittest/main.py", line 149, in parseArgs
    self.createTests()
  File "/usr/lib/python2.7/unittest/main.py", line 158, in createTests
    self.module)
  File "/usr/lib/python2.7/unittest/loader.py", line 130, in loadTestsFromNames
    suites = [self.loadTestsFromName(name, module) for name in names]
  File "/usr/lib/python2.7/unittest/loader.py", line 103, in loadTestsFromName
    return self.loadTestsFromModule(obj)
  File "build/bdist.linux-x86_64/egg/setuptools/command/test.py", line 39, in␣
↪loadTestsFromModule
  File "/usr/lib/python2.7/unittest/loader.py", line 100, in loadTestsFromName
    parent, obj = obj, getattr(obj, part)
AttributeError: 'module' object has no attribute 'contention_detecting_plugin'
```

The error, `AttributeError: 'module' object has no attribute 'xxx'`, is not very helpful in identifying the source of the error. Running pylint (as below) however, will give an error output in the following form:

```
[
    {
        "message": "Unable to import 'NOTabc'",
        "obj": "",
        "column": 0,
        "path": "src/metaswitch/clearwater/cluster_manager/plugin_base.py",
        "line": 33,
        "type": "error",
        "symbol": "import-error",
        "module": "metaswitch.clearwater.cluster_manager.plugin_base"
    }
]
```

The following commands will install and run pylint in your local virtual environment:

```
make clean && make env
_env/bin/easy_install pylint
PYTHONPATH=src:common _env/bin/python -m pylint --disable=all --enable=F,E --output-
↪format json <path_to_modules>
```

Where the `path_to_modules` is the module or package directory of the code you want to check. For our projects, this will usually be under `src/metaswitch/`, e.g. in the clearwater-etcd repo, to test all of clearwater-etcd, use `src/metaswitch/clearwater/` as the path, but to have pylint inspect just the cluster-manager plugin_base file, use `src/metaswitch/clearwater/cluster_manager/plugin_base.py`.

Stress Testing

One of Clearwater's biggest strengths is scalability and in order to demonstrate this, we have easy-to-use settings for running large amounts of SIP stress against a deployment. This document describes:

- Clearwater's SIP stress nodes, what they do, and (briefly) how they work
- how to kick off your own stress test.

SIP Stress Nodes

A Clearwater SIP stress node is similar to any other Project Clearwater node, except that instead of having a Debian package like `bono` or `sprout` installed, it has our `clearwater-sip-stress-coreonly` or `clearwater-sip-stress` Debian package installed.

What they do

Clearwater SIP stress nodes provide a set of SIPp scripts to run against your Sprout cluster. There are two kinds of stress available:

- Our recommended approach is to use scripts which emulate a P-CSCF and send traffic to Sprout, stress testing the IMS Core directly. The nodes log their success/failure to `/var/log/clearwater-sip-stress` and also print a human-readable summary of the stress (with information about percentage of failed calls, average latency, and so on).
- We also have some older scripts which emulate UEs and send traffic to Bono, stress testing the P-CSCF and the IMS Core together. The nodes log their success/failure to `/var/log/clearwater-sipp`, but unlike the other scripts, do not print a human-readable summary of the stress.

Deploying a stress node

These instructions assume you've already installed a Clearwater deployment, either manually or through Chef.

Follow this process to bulk provision the number of subscribers you want. As a general guideline, we'd expect a small deployment (with one Sprout, Vellum and Dime, each with one full CPU core) to handle at least 30,000 subscribers.

You should also ensure that the `reg_max_expires` setting is set to 1800 rather than the default of 300 - see our configuration guide for instructions. This matches the assumptions in our load profile of 2 re-registers per hour.

Chef

If you're using Chef to automate your Clearwater install, create your stress test node by typing `knife box create -V -E ENVIRONMENT sipp --index 1`.

Manual install

Otherwise, follow these steps to deploy a stress node manually:

- create a new virtual machine and bootstrap it by configuring access to the Project Clearwater Debian repository.
- set the following property in `/etc/clearwater/local_config`:
  - `local_ip` - the local IP address of this node
- for our recommended scripts which send stress to Sprout, run `sudo apt-get install clearwater-sip-stress-coreonly` to install the Debian packages.
- for our older scripts which send stress to Bono, run `sudo apt-get install clearwater-sip-stress` to install the Debian packages.

The stress node needs to be able to open connections to Sprout on TCP ports 5052 and 5054, and Sprout needs to be able to open connections to the stress node on TCP port 5082.

Running stress (IMS core only)

To kick off a stress run, simply run: `/usr/share/clearwater/bin/run_stress <home_domain> <number of subscribers> <duration in minutes>`.

The script will then:

- set up the stress test by sending an initial register for all the subscribers
- report that that initial registration has completed and that the stress test is starting
- send traffic, using a fixed load profile of 1.3 calls/hour for each subscriber (split equally between incoming and outgoing calls) and 2 re-registrations per hour
- after the given duration, wait for all calls to finish and then exit
- print summary output about how the test went

The output will be in this format:

```
Starting initial registration, will take 6 seconds
Initial registration succeeded
Starting test
Test complete

Elapsed time: 00:15:07
```

```
Start: 2016-11-16 15:03:29.062755
End: 2016-11-16 15:18:36.414653

Total calls: 1625
Successful calls: 1612 (99.2%)
Failed calls: 13 (0.8%)

Retransmissions: 0

Average time from INVITE to 180 Ringing: 74.0 ms
# of calls with 0-2ms from INVITE to 180 Ringing: 0 (0.0%)
# of calls with 2-20ms from INVITE to 180 Ringing: 0 (0.0%)
# of calls with 20-200ms from INVITE to 180 Ringing: 1572 (96.7384615385%)
# of calls with 200-2000ms from INVITE to 180 Ringing: 45 (2.76923076923%)
# of calls with 2000+ms from INVITE to 180 Ringing: 0 (0.0%)

Total re-REGISTERs: 1800
Successful re-REGISTERs: 1792 (99.5555555556%)
Failed re-REGISTERS: 8 (0.444444444444%)

REGISTER retransmissions: 0

Average time from REGISTER to 200 OK: 15.0 ms

Log files in /var/log/clearwater-sip-stress
```

The summary statistics at the end just come from SIPp, not the Clearwater deployment. If you want to see our more detailed statistics, you'll need to be running a separate monitoring tool such as Cacti (and we provide the start and end time of the stress run, to let you match up with these external graphs).

Extra run_stress options

The run-stress script has some command-line options:

- `--initial-reg-rate` - this controls how many REGISTERs/second the script sends in during the initial registration phase (defaulting to 80). On systems that can cope with the load, raising this value will let the test run start faster.
- `--sipp-output` - By default, the script hides the SIPp output and just presents the end-of-run stats. With this option, it will show the SIPp output screen, which may be useful for users familiar with SIPp.
- `--icscf-target TARGET` - Domain/IP and port to target registration stress at. Default is `sprout.{domain}:5052`.
- `--scscf-target TARGET` - Domain/IP and port to target call stress at. Default is `sprout.{domain}:5054`.

Running stress (IMS core and P-CSCF)

In this mode, each SIP stress node picks a single bono to generate traffic against. This bono is chosen by matching the bono node's index against the SIP stress node's index.

This test includes two important scripts.

- `/usr/share/clearwater/infrastructure/scripts/sip-stress`, which generates a `/usr/share/clearwater/sip-stress/users.csv.1` file containing the list of all subscribers we should be targeting - these are calculated from properties in `/etc/clearwater/shared_config`.

- `/etc/init.d/clearwater-sip-stress`, which runs `/usr/share/clearwater/bin/sip-stress`, which in turn runs SIPp specifying `/usr/share/clearwater/sip-stress/sip-stress.xml` as its test script. This test script simulates a pair of subscribers registering every 5 minutes and then making a call every 30 minutes.

The stress test logs to `/var/log/clearwater-sip-stress/sipp.<index>.out`.

There is some extra configuration needed in this mode, so you should:

- set the following properties in `/etc/clearwater/shared_config`:
  - (required) `home_domain` - the home domain of the deployment under test
  - (optional) `bono_servers` - a list of bono servers in this deployment
  - (optional) `stress_target` - the target host (defaults to the `node_idx`-th entry in `bono_servers` or, if there are no `bono_servers`, defaults to `home_domain`)
  - (optional) `base` - the base directory number (defaults to 2010000000)
  - (optional) `count` - the number of subscribers to run on this node (must be even, defaults to 30000)
- optionally, set the following property in `/etc/clearwater/local_config`:
  - node_idx - the node index (defaults to 1)

To apply this config and start stress, run `sudo /usr/share/clearwater/infrastructure/scripts/sip-stress` and `sudo service clearwater-sip-stress restart`.

Using Cacti with Clearwater

Cacti is an open-source statistics and graphing solution. It supports gathering statistics via native SNMP and also via external scripts. It then exposes graphs over a web interface.

We use Cacti for monitoring Clearwater deployment, in particular large ones running stress.

This document describes how to

- create and configure a Cacti node
- point it at your Clearwater nodes
- view graphs of statistics from your Clearwater nodes.

Setting up a Cacti node (automated)

Assuming you've followed the Automated Chef install, here are the steps to create and configure a Cacti node:

1. use knife box create to create a Cacti node - `knife box create -E <name> cacti`
2. set up a DNS entry for it - `knife dns record create -E <name> cacti -z <root> -T A --public cacti -p <name>`
3. create graphs for all existing nodes by running `knife cacti update -E <name>`
4. point your web browser at `cacti.<name>.<root>/cacti/` (you may need to wait for the DNS entry to propagate before this step works)

If you subsequently scale up, running `knife cacti update -E <name>` again will create graphs for the new nodes (without affecting the existing nodes).

Setting up a Cacti node (manual)

If you haven't followed our automated install process, and instead just have an Ubuntu 14.04 machine you want to use to monitor Clearwater, then:

1. install Cacti on a node by running `sudo apt-get install cacti cacti-spine`
2. accept all the configuration defaults
3. login to the web UI at `http://<IP address>/cacti` (using the credentials admin/admin) and set a new password
4. modify configuration by
   (a) going to Devices and deleting "localhost"
   (b) going to Settings->Poller and set "Poller Type" to "spine" and "Poller Interval" to "Every Minute" - then click Save at the bottom of the page
   (c) going to Data Templates and, for each of "ucd/net - CPU Usage - *" set "Associated RRA's" to "Hourly (1 Minute Average)" and "Step" to 60
   (d) going to Graph Templates and change "ucd/net - CPU Usage" to disable "Auto Scale"
   (e) going to Import Templates and import the Sprout, Bono and SIPp host templates - these define host templates for retrieving statistics from our components via SNMP and 0MQ. For each template you import, select "Select your RRA settings below" and "Hourly (1 Minute Average)"
   (f) set up the 0MQ-querying script by
      i. ssh-ing into the cacti node
      ii. running the following

```
sudo apt-get install -y --force-yes git ruby1.9.3 build-essential libzmq3-
↪dev
sudo gem install bundler --no-ri --no-rdoc
git clone https://github.com/Metaswitch/cpp-common.git
cd cpp-common/scripts/stats
sudo bundle install
```

Pointing Cacti at a Node

Before you point Cacti at a node, make sure the node has the required packages installed. All nodes need clearwater-snmpd installed (`sudo apt-get install clearwater-snmpd`). Additionally, sipp nodes need clearwater-sip-stress-stats (`sudo apt-get install clearwater-sip-stress-stats`).

To manually point Cacti at a new node,

1. go to Devices and Add a new node, giving a Description and Hostname, setting a Host Template of "Bono", "Sprout" or "SIPp" depending on the node type), Downed Device Detection to "SNMP Uptime" and SNMP Community to "clearwater"
2. click "Create Graphs for this Host" and select the graphs that you want - "ucd/net - CPU Usage" is a safe bet, but you might also want "Client Counts" and "Bono Latency" (if a bono node), "Sprout Latency" (if a sprout node), or "SIP Stress Status" (if a sipp node).
3. click "Edit this Host" to go back to the device, choose "List Graphs", select the new graphs and choose "Add to Default Tree" - this exposes them on the "graphs" tab you can see at the top of the page (although it may take a couple of minutes for them to accumulate enough state to render properly).

Viewing Graphs

Graphs can be viewed on the top "graphs" tab. Useful features include

- the "Half hour" preset, which shows only the last half-hour rather than the last day
- the search box, which matches on graph name
- the thumbnail view checkbox, which gets you many more graphs on screen
- the auto-refresh interval, configured on the settings panel (the default is 5 minutes, so if you're looking for a more responsive UI, you'll need to set a smaller value)
- CSV export, achievable via the icon on the right of the graph.

Running the Clearwater Live Tests

The Clearwater team have put together a suite of live tests that can be run over a deployment to confirm that the high level function is working. These instructions will take you through installing the test suite and running the tests.

Prerequisites

- You've installed Clearwater
- You have an Ubuntu Linux machine available to drive the tests
- If you installed through the automated install process, the chef workstation is a perfectly good choice for this task

Installing dependencies

On your test machine, run

```
sudo apt-get install build-essential git --yes
curl -L https://get.rvm.io | bash -s stable
source ~/.rvm/scripts/rvm
rvm autolibs enable
rvm install 1.9.3
rvm use 1.9.3
```

This will install Ruby version 1.9.3 and its dependencies.

Fetching the code

Run the following to download and install the Clearwater test suite

```
git clone -b stable --recursive git@github.com:Metaswitch/clearwater-live-test.git
cd clearwater-live-test
bundle install
```

Make sure that you have an SSH key - if not, see the GitHub instructions for how to create one.

Work out your signup code

The tests need your signup code to create a test user. You set this as `signup_key` during install: manually in /etc/clearwater/shared_config or automatically in knife.rb. For the rest of these instructions, the signup code will be referred to as `<code>`.

Running the tests against an All-in-One node

Work out your All-in-One node's identity

This step is only required if you installed an All-in-One node, either from an AMI or an OVF. If you installed manually or using the automated install process, just move on to the next step.

If you installed an All-in-One node from an Amazon AMI, you need the public DNS name that EC2 has assigned to your node. This will look something like `ec2-12-34-56-78.compute-1.amazonaws.com` and can be found on the EC2 Dashboard on the "instances" panel. If you installed an All-in-One node from an OVF image, you need the IP address that was assigned to the node via DHCP. You can find this out by logging into the node's console and typing `hostname -I`.

For the rest of these instructions, the All-in-One node's identity will be referred to as `<aio-identity>`.

Run the tests

To run the subset of the tests that don't require PSTN interconnect to be configured, simply run

```
rake test[example.com] SIGNUP_CODE=<code> PROXY=<aio-identity> ELLIS=<aio-identity>
```

The suite of tests will be run and the results will be printed on-screen.

Running the tests against a full deployment

Work out your base domain

- If you installed Clearwater manually, your base DNS name will simply be `<zone>`.
- If you installed using the automated install process, your base DNS name will be `<name>.<zone>`.

For the rest of these instructions, the base DNS name will be referred to as `<domain>`.

Running the tests

To run the subset of the tests that don't require PSTN interconnect to be configured, simply run

```
rake test[<domain>] SIGNUP_CODE=<code>
```

The suite of tests will be run and the results will be printed on-screen. If your deployment doesn't have DNS entries for the Bono and Ellis domain, then these can be passed to rake by adding the `PROXY` and `ELLIS` parameters, e.g.

```
rake test[<domain>] SIGNUP_CODE=<code> PROXY=<Bono domain> ELLIS=<Ellis domain>
```

You can find the full list of parameters for the tests here.

PSTN testing

The live test framework also has the ability to test various aspects of PSTN interconnect. If you've configured your Clearwater deployment with an IBCF node that connects it to a PSTN trunk, you can test that functionality by picking a real phone (e.g. your mobile) and running

```
rake test[<domain>] PSTN=true LIVENUMBER=<your phone #> SIGNUP_CODE=<code>
```

Which will test call services related to the PSTN and will ring your phone and play an audio clip to you (twice, to cover both TCP and UDP).

Where next?

Now that you've confirmed that your Clearwater system is operational, you might want to make a real call or explore Clearwater to see what else Clearwater offers.