blockade Documentation

Release 0.3.1

Quest

May 22, 2017

Contents

Blockade is a utility for testing network failures and partitions in distributed applications. Blockade uses Docker containers to run application processes and manages the network from the host system to create various failure scenarios.

A common use is to run a distributed application such as a database or cluster and create network partitions, then observe the behavior of the nodes. For example in a leader election system, you could partition the leader away from the other nodes and ensure that the leader steps down and that another node emerges as leader.

Blockade features:

- A flexible YAML format to describe the containers in your application
- Support for dependencies between containers, using named links
- A CLI tool for managing and querying the status of your blockade
- When run as a daemon, a simple *REST API* can be used to configure your blockade
- Creation of arbitrary partitions between containers
- Giving a container a flaky network connection to others (drop packets)
- Giving a container a slow network connection to others (latency)
- While under partition or network failure control, containers can freely communicate with the host system – so you can still grab logs and monitor the application.

Blockade was originally developed by the Dell Cloud Manager (formerly Enstratius) team. Blockade is inspired by the excellent Jepsen article series.

Get started with the *Blockade Guide*!

CHAPTER 1

Reference Documentation

Requirements

You need an accessible Docker Engine API, and the ability to launch privileged containers with host networking. Docker can be local or remote. If remote, set `DOCKER_HOST` and the other environment variables to configure the URL and credentials. Generally, if the `docker` cli works, so should Blockade.

Docker Swarm is not supported at this time.

Installing

Blockade can be installed via `pip` or `easy_install`:

```
$ pip install blockade
```

macOS or Windows

Blockade works on macOS either natively pointing to a remote Docker Engine API or via Docker for Mac.

Blockade does not support Windows native containers. Nor is it known to work with Docker for Windows, but this may be possible. One option is to run Blockade itself in a container, in daemon mode, and talk to it via the *REST API*.

Another great option is Vagrant, to run Blockade and Docker in a Linux VM. Use the included `Vagrantfile` or another approach to get Docker and Blockade installed into a Linux VM. If you have Vagrant installed, running `vagrant up` from the Blockade checkout directory should get you started. Note that this may take a while, to download needed VMs and Docker containers.

Blockade Guide

This guide walks you through a simple example that highlights the power of Blockade. We will start a fake "application" consisting of three Docker containers. The first runs a simple `sleep` command. The other two containers ping the first. With this simple structure, we can easily see what happens when we introduce partitions and network failures between the containers.

Check your Blockade install

To check your install, run the following commands:

```
# check docker
$ docker info

# check blockade
$ blockade -h
```

If you get an error from either command, you'll need to fix this before proceeding. See the Docker installation docs and *Requirements*.

Set up your Blockade config

Now create a new directory and in it create a `blockade.yaml` file with these contents:

```
containers:
  c1:
    image: ubuntu:trusty
    command: /bin/sleep 300000
    ports: [10000]

  c2:
    image: ubuntu:trusty
    command: sh -c "ping $C1_PORT_10000_TCP_ADDR"
    links: ["c1"]

  c3:
    image: ubuntu:trusty
    command: sh -c "ping $C1_PORT_10000_TCP_ADDR"
    links: ["c1"]
```

This configuration specifies the three containers we described above. Note that we rely on Docker named links which require at least one open port. Hence our sleeping `c1` container has a fake port 10000 open. The `ubuntu:trusty` image must exist in your Docker installation. You can download it using the docker pull command `sudo docker pull ubuntu:trusty`.

Start the Blockade

Now use the `blockade up` command to stand up our containers:

```
$ blockade up

NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
c1              b9794aaeed42    UP      172.17.0.2      NORMAL
```

```
c2              875885f54593    UP      172.17.0.4      NORMAL
c3              9b7227b42466    UP      172.17.0.3      NORMAL
```

You should see output like above. Note that you get the local IP address and Docker container ID for each container.

Now let's take a look at the output of `c2`, which is pinging `c1`. We'll use the `blockade logs` command, but pipe it through tail so we just get the last several lines:

```
$ blockade logs c2 | tail
64 bytes from 172.17.0.2: icmp_req=59 ttl=64 time=0.067 ms
64 bytes from 172.17.0.2: icmp_req=60 ttl=64 time=0.077 ms
64 bytes from 172.17.0.2: icmp_req=61 ttl=64 time=0.077 ms
64 bytes from 172.17.0.2: icmp_req=62 ttl=64 time=0.073 ms
64 bytes from 172.17.0.2: icmp_req=63 ttl=64 time=0.076 ms
64 bytes from 172.17.0.2: icmp_req=64 ttl=64 time=0.070 ms
64 bytes from 172.17.0.2: icmp_req=65 ttl=64 time=0.078 ms
64 bytes from 172.17.0.2: icmp_req=66 ttl=64 time=0.073 ms
64 bytes from 172.17.0.2: icmp_req=67 ttl=64 time=0.109 ms
```

The `blockade logs` command is the same as the `docker logs` command, it grabs any stderr and or stdout output from the container.

Mess with the network

Now let's try a couple network filters. We'll make the network to `c2` be slow and the network to `c3` be flaky.

```
$ blockade slow c2

$ blockade flaky c3

$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
c1              b9794aaeed42    UP      172.17.0.2      NORMAL
c2              875885f54593    UP      172.17.0.4      SLOW
c3              9b7227b42466    UP      172.17.0.3      FLAKY
```

Now look at the logs for `c2` and `c3` again:

```
$ blockade logs c2 | tail
64 bytes from 172.17.0.2: icmp_req=358 ttl=64 time=126 ms
64 bytes from 172.17.0.2: icmp_req=359 ttl=64 time=0.077 ms
64 bytes from 172.17.0.2: icmp_req=360 ttl=64 time=64.5 ms
64 bytes from 172.17.0.2: icmp_req=361 ttl=64 time=265 ms
64 bytes from 172.17.0.2: icmp_req=362 ttl=64 time=158 ms
64 bytes from 172.17.0.2: icmp_req=363 ttl=64 time=64.8 ms
64 bytes from 172.17.0.2: icmp_req=364 ttl=64 time=3.47 ms
64 bytes from 172.17.0.2: icmp_req=365 ttl=64 time=90.2 ms
64 bytes from 172.17.0.2: icmp_req=366 ttl=64 time=0.067 ms

$ blockade logs c3 | tail
64 bytes from 172.17.0.2: icmp_req=415 ttl=64 time=0.075 ms
64 bytes from 172.17.0.2: icmp_req=416 ttl=64 time=0.079 ms
64 bytes from 172.17.0.2: icmp_req=419 ttl=64 time=0.063 ms
64 bytes from 172.17.0.2: icmp_req=420 ttl=64 time=0.065 ms
64 bytes from 172.17.0.2: icmp_req=421 ttl=64 time=0.063 ms
64 bytes from 172.17.0.2: icmp_req=425 ttl=64 time=0.062 ms
64 bytes from 172.17.0.2: icmp_req=426 ttl=64 time=0.079 ms
```

```
64 bytes from 172.17.0.2: icmp_req=427 ttl=64 time=0.056 ms
64 bytes from 172.17.0.2: icmp_req=428 ttl=64 time=0.066 ms
```

Note how the time value of the `c2` pings is erratic, while `c3` is missing many packets (look at the `icmp_req` value – it should be sequential).

Now let's use `blockade fast` to fix the network:

```
$ blockade fast --all

$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
c1              6367a903f093    UP      172.17.0.2      NORMAL
c2              35efaf92bba0    UP      172.17.0.4      NORMAL
c3              e8ed611a38de    UP      172.17.0.3      NORMAL
```

Partition the network

Blockade can also create partitions between containers. This is valuable for testing split-brain behaviors. To demonstrate, let's partition node `c2` off from the other two containers. It will no longer be able to ping `c1`, but `c3` will continue unhindered.

Partitions are specified as groups of comma-separated container names:

```
$ blockade partition c1,c3 c2

$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
c1              6367a903f093    UP      172.17.0.2      NORMAL     1
c2              35efaf92bba0    UP      172.17.0.4      NORMAL     2
c3              e8ed611a38de    UP      172.17.0.3      NORMAL     1
```

Note the partition column: `c1` and `c3` are in partition #1 while `c2` is in partition #2.

You can now use `blockade logs` to check the output of `c2` and `c3` and see the partition in effect.

Restore the network with the `join` command:

```
$ blockade join
$ blockade status
NODE            CONTAINER ID    STATUS  IP              NETWORK    PARTITION
c1              6367a903f093    UP      172.17.0.2      NORMAL
c2              35efaf92bba0    UP      172.17.0.4      NORMAL
c3              e8ed611a38de    UP      172.17.0.3      NORMAL
```

Tear down the Blockade

Once finished, kill the containers and restore the network with the `destroy` command:

```
$ blockade destroy
```

Next steps

Next, check out the reference details in *Configuration* and *Commands*.

Configuration

The blockade configuration file is conventionally named `blockade.yaml` and is used to describe the containers in your application. Here is an example:

```
containers:
  c1:
    image: my_docker_image
    command: /bin/myapp
    volumes: {"/opt/myapp": "/opt/myapp_host"}
    expose: [80]
    ports: {8080: 80}
    environment: {"IS_MASTER": 1}

  c2:
    image: my_docker_image
    command: /bin/myapp
    volumes: ["/data"]
    links: {c1: master}

  c3:
    image: my_docker_image
    command: /bin/myapp
    links: {c1: master}

network:
  flaky: 30%
  slow: 75ms 100ms distribution normal
```

The format is YAML and there are two important sections: `containers` and `network`.

Containers

Containers are described as a map with the key as the Blockade container name (`c1`, `c2`, `c3` in the example above). This key is used for commands to manipulate the Blockade and is also used as the hostname of the container.

Each entry in the `containers` section is a single Docker container in the Blockade. Each container parameter controls how the container is launched. Most are simply pass-throughs to Docker. Many valuable details can be found in the Docker run command documentation.

image

`image` is required and specifies the Docker image name to use for the container. The image must exist in your Docker installation.

command

`command` is optional and specifies the command to run within the container. If not specified, a default command must be part of the image you are using. You may include environment variables in this command, but to do so you must typically wrap the command in a shell, like `sh -c "/bin/myapp $MYENV"`.

volumes

`volumes` is optional and specifies the volumes to mount in the container, *from the host*. Volumes can be specified as either a map or a list. In map form, the key is the path *on the host* to expose and the value is the mountpoint *within the container*. In list form, the host path and container mountpoint are assumed to be the same. See the Docker volumes documentation for details about how this works.

expose

`expose` is optional and specifies ports to expose from the container. Ports must be exposed in order to use the Docker named links feature.

links

`links` is optional and specifies links from one container to another. A dependent container will be given environment variables with the parent container's IP address and port information. See named links documentation for details.

ports

`ports` is optional and specifies ports published to the host machine. It is a dictionary from external port to internal container port.

environment

`environment` is optional and specifies environment variables for `command`. See more details in `command` section above.

hostname

`hostname` is optional and gives the ability to redefine hostname of a container.

dns

`dns` is optional and specifies a list of DNS-servers for container.

start_delay

`start_delay` is optional and specifies a number of seconds to wait before starting a container. This can be used as a stopgap way to ensure a dependent service is running before starting a container.

count

`count` is optional and specifies the number of copies of the container to launch.

cap_add

`cap_add` is optional and specifies additional root capabilities

container_name

`container_name` is optional and specifies a custom container name, instead of letting blockade generate one. Use caution with this setting, because Docker enforces uniqueness of names across all containers.

When this parameter is combined with `count`, an underscore and index will be suffixed to this name. For example "app" becomes "app_1", "app_2", etc.

Network

The `network` configuration block controls the settings used for network filter commands like `slow` and `flaky`. If unspecified, defaults will be used. There are two parameters:

slow

`slow` controls the amount and distribution of delay for network packets when a container is in the Blockade slow state. It is specified as an expression understood by the tc netem traffic control `delay` facility. See the man page for details, but the pattern is:

```
TIME [ JITTER [ CORRELATION ] ]
    [ distribution { uniform | normal | pareto |  paretonormal } ]
```

`TIME` and `JITTER` are expressed in milliseconds while `CORRELATION` is a percentage.

flaky

`flaky` controls the lossiness of network packets when a contrainer is in the Blockade flaky state. It is specified as an expression understood by the tc netem traffic control `loss` facility. See the man page for details, but the simplified pattern is:

```
random PERCENT [ CORRELATION ]
```

`PERCENT` and `CORRELATION` are both expressed as percentages.

driver

`driver` specifies docker network stack. `default` will use standard Docker networking, that allows to connect containers by links. Other option is `udn`. It will enable user defined network, that performs dns resolution of running containers and allows to create any-to-any communications. In case of `udn` network environment variables with links will not bet set.

Commands

The Blockade CLI is built to make it easy to manually manage your containers, and is also easy to wrap in scripts as needed. All commands that produce output support a `--json` flag to output in JSON instead of plain text.

For the most up to date and detailed command help, use the built-in CLI help system (`blockade --help`).

up

```
usage: blockade up [--json]

Start the containers and link them together

  --json      Output in JSON format
```

destroy

```
usage: blockade destroy

Destroy all containers and restore networks
```

status

```
usage: blockade status [--json]

Print status of containers and networks

optional arguments:
  --json      Output in JSON format
```

start

```
usage: blockade start [--all] [CONTAINER [CONTAINER ...]]

Start some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

stop

```
usage: blockade stop [--all] [CONTAINER [CONTAINER ...]]

Stop some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

kill

```
usage: blockade kill [--all] [--signal] [CONTAINER [CONTAINER ...]]

Kill some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container

optional arguments:

  --signal      Specify the signal to be sent (str or int). Defaults to SIGKILL.
```

restart

```
usage: blockade restart [--all] [CONTAINER [CONTAINER ...]]

Restart some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

logs

```
usage: blockade logs CONTAINER

Fetch the logs of a container

  CONTAINER    Container to fetch logs for
```

flaky

```
usage: blockade flaky [--all] [CONTAINER [CONTAINER ...]]

Make the network flaky for some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

duplicate

```
usage: blockade duplicate [--all] [CONTAINER [CONTAINER ...]]

Introduce packet duplication into the network of some or all containers
```

```
  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

slow

```
usage: blockade slow [--all] [CONTAINER [CONTAINER ...]]

Make the network slow for some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

fast

```
usage: blockade fast [--all] [CONTAINER [CONTAINER ...]]

Restore network speed and reliability for some or all containers

  CONTAINER   Container to select

  --all       Select all containers
  --random    Select a random container
```

partition

```
usage: blockade partition [--random] [PARTITION [PARTITION ...]]

Partition the network between containers

    Replaces any existing partitions outright. Any containers NOT specified
    in arguments will be globbed into a single implicit partition. For
    example if you have three containers: c1, c2, and c3 and you run:

        blockade partition c1

    The result will be a partition with just c1 and another partition with
    c2 and c3.

    Alternatively, ``--random`` may be specified, and zero or more random
    partitions will be generated by blockade.


  PARTITION   Comma-separated partition

  --random    Randomly select zero or more partitions of containers
```

join

```
usage: blockade join

Restore full networking between containers
```

add

```
usage: blockade add [CONTAINER [CONTAINER ...]]

Add existing Docker containers to a Blockade

  CONTAINER   Container to add
```

Changelog

0.3.1 (2016-12-09)

- #43: Restore support for loading from `blockade.yml` config file.
- #26: Improved error messages when running blockade without access to the Docker API.
- #25: Improved error messages when determining container host network device fails.
- #40: Fixed `kill` command (broken in 0.3.0).
- #1: Fixed support for configuring Docker API via `DOCKER_HOST` env.
- #36: Truncate long blockade IDs to avoid iptables limits.
- Switched to directly inspecting `/sys` for container network devices instead of via `ip`. This means containers no longer need to have `ip` installed.
- Improved Blockade Python API by returning names of the containers a command has operated on. Contributed by Gregor Uhlenheuer (@kongo2002).
- Fixed `Vagrantfile` to also work on Windows. Contributed by Oresztesz Margaritisz (@gitaroktato).
- Documentation fix contributed by Konrad Klocek (@kklocek).
- Added new `version` command that prints Blockade version and exits.
- Added `cap_add` container config option, for specifying additional root capabilities. Contributed by Maciej Zimnoch (@Zimnx).

0.3.0 (2016-10-29)

- Reworks all network commands to run in Docker containers. This allows Blockade to be run without root privileges, as long as the user can access Docker.
- Introduces a REST API and daemon mode that allows creation and management of blockades remotely.
- Adds ability to add a container to a running blockade, via `add` command.
- Adds support for Docker user-defined networks to allow any-to-any communication between containers, without links. Contributed by Stas Kelvich (@kelvich).

- Adds ability to configure DNS servers for containers in a blockade. Contributed by Vladimir Borodin (@dev1ant).
- Adds a generic `--random` flag for many commands to allow easier randomized chaos testing. Contributed by Gregor Uhlenheuer (@kongo2002).
- Introduces a new `kill` command for killing containers in a blockade.
- Fixed links to Docker documentation. Contributed by @joepadmiraal.
- Fixed links of named containers. Contributed by Gregor Uhlenheuer (@kongo2002).

0.2.0 (2015-12-23)

- #14: Support for docker >1.6, with the native driver. Eliminates the need to use the deprecated LXC driver. Contributed by Gregor Uhlenheuer.
- #12: Fix port publishing. **Breaking change**: the order of port publishing was swapped to be `{external: internal}`, to be consistent with the docker command line. Contributed by aidanhs.
- Introduces new `duplicate` command, which causes some packets to a container to be duplicated. Contributed by Gregor Uhlenheuer.
- Introduces new `start`, `stop`, and `restart` commands, which manage specified containers via Docker. Contributed By Gregor Uhlenheuer.
- Introduces new random partition behavior: `blockade partition --random` will create zero or more random partitions. Contributed By Gregor Uhlenheuer.
- Reworked the blockade ID generation to be more like docker-compose, instead of using randomly-generated IDs. If `--name` is specified on the command line, this is used as the blockade ID and is prefixed to container names. Otherwise the blockade name is taken from the basename of the current working directory.
- Numerous other small fixes and features, many contributed by Gregor Uhlenheuer. Thanks Gregor!

0.1.2 (2015-1-28)

- #6: Change `ports` config keyword to match docker usage. It now publishes a container port to the host. The `expose` config keyword now offers the previous behavior of `ports`: it makes a port available from the container, for linking to other containers. Thanks to Simon Bahuchet for the contribution.
- #9: Fix logs command for Python 3.
- Updated dependencies.

0.1.1 (2014-02-12)

- Support for Python 2.6 and Python 3.x

0.1.0 (2014-02-11)

- Initial release of Blockade!

REST API

The REST API is provided by the Blockade daemon and exposes most of the Blockade commands. In particular, it can be helpful for automated test suites, allowing them to setup, manipulate, and destroy Blockades through the API.

Check the help for Blockade daemon options `blockade daemon -h`

Create a Blockade

Example request:

```
POST /blockade/<name>
Content-Type: application/json

{
    "containers": {
        "c1": {
            "image": "ubuntu:trusty",
            "hostname": "c1",
            "command": "/bin/sleep 300"
        },
        "c2": {
            "image": "ubuntu:trusty",
            "hostname": "c2",
            "command": "/bin/sleep 300"
        }
    }
}
```

Response:

```
204 No content
```

Execute an action on a Blockade (start, stop, restart, kill)

Example request:

```
POST /blockade/<name>/action
Content-Type: application/json

{
    "command": "start",
    "container_names": ["c1"]
}
```

Response:

```
204 No content
```

Change the network state of a Blockade (fast, slow, duplicate, flaky)

Example request:

```
POST /blockade/<name>/network_state
Content-Type: application/json

{
    "network_state": "fast",
    "container_names": ["c1"]
}
```

Response:

```
204 No content
```

Partition the network between containers

Example request:

```
POST /blockade/<name>/partitions
Content-Type: application/json

{
    "partitions": [["c1"], ["c2", "c3"]]
}
```

Response:

```
204 No content
```

Delete all partitions for a Blockade and restore full connectivity

Example request:

```
DELETE /blockade/<name>/partitions
```

Response:

```
204 No content
```

List all Blockades

Example request:

```
GET /blockade
```

Response:

```
{
    "blockades": [
        "test_blockade1",
        "test_blockade2"
    ]
}
```

Get Blockade

Example request:

```
GET /blockade/<name>
```

Response:

```
{
    "containers": {
        "c1": {
            "container_id":
↪"729a67bc126f597b563410b8b5478929da04ba81c0ce4519c2d7eb48599a4406",
            "device": "veth035b534",
            "ip_address": "172.17.0.7",
            "name": "c1",
            "network_state": "NORMAL",
            "partition": null,
            "status": "UP"
        },
        "c2": {
            "container_id":
↪"ee84117d7b6fd806279ee0e5a2a3737a8d21a1e5129df31d3e0f1dee22d94d35",
            "device": "veth304bac6",
            "ip_address": "172.17.0.6",
            "name": "c2",
            "network_state": "NORMAL",
            "partition": null,
            "status": "UP"
        }
    }
}
```

Add an existing Docker container to a Blockade

Example request:

```
PUT /blockade/<name>
Content-Type: application/json

{
    "containers": ["docker_container_id"]
}
```

Response:

```
204 No content
```

Delete a Blockade

Example request:

```
DELETE /blockade/<name>
```

Response:

```
204 No content
```

Chaos REST API

Users wishing to start *chaos* on their blockade can use this REST API. Based on the parameters given the *chaos* feature will randomly select containers in the blockade to perform blockade events (duplicate, slow, flaky, or partition) upon.

Start chaos on a Blockade

Began performing chaos operations on a given blockade. The user can control the number of containers that can be effected in a given degradation period as well as what possible events can be selected. A *degradation* period will start sometime between *min_start_delay* and *max_start_delay* milliseconds and it will last for between *min_run_time* and *max_run_time* milliseconds.

Example request:

```
POST /blockade/<name>/chaos
Content-Type: application/json

{
    "min_start_delay": 30000,
    "max_start_delay": 300000,
    "min_run_time": 30000,
    "max_run_time": 300000,
    "min_containers_at_once": 1,
    "max_containers_at_once": 2,
    "event_set": ["SLOW", "DUPLICATE", "FLAKY", "STOP", "PARTITION"]
}
```

Response:

```
201 Successfully started chaos on <name>
```

Update chaos parameters on a Blockade

This operation takes the same options as the create.

Example request:

```
POST /blockade/<name>/chaos
Content-Type: application/json

{
    "min_start_delay": 30000,
    "max_start_delay": 300000,
    "min_run_time": 30000,
    "max_run_time": 300000,
    "min_containers_at_once": 1,
    "max_containers_at_once": 2,
    "min_events_at_once": 1,
    "max_events_at_once": 2,
    "event_set": ["SLOW", "DUPLICATE", "FLAKY", "STOP", "PARTITION"]
}
```

Response:

```
200 Updated chaos on <name>
```

Get the current status of chaos

Example request:

```
GET /blockade/<name>/chaos
```

Response:

```
{
    "state": "DEGRADED"
}
```

Stop chaos on a give blockade

Example request:

```
DELETE /blockade/<name>/chaos
```

Response:

```
Deleted chaos on <name>
```

CHAPTER 2

Development and Support

Blockade is available on github. Bug reports should be reported as issues there.

CHAPTER 3

License

Blockade is offered under the Apache License 2.0.