antidote Documentation

Matt Oswalt

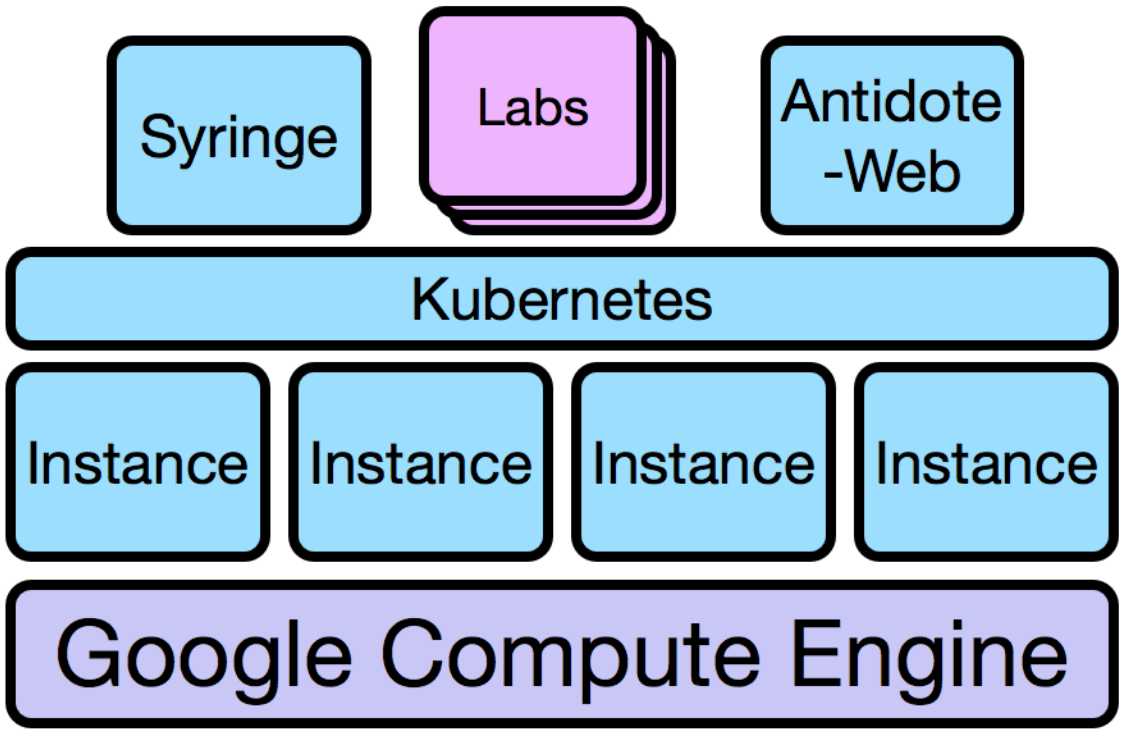
Feb 26, 2020

Contents:

CHAPTER 1

Architecture

Antidote has multiple layers that build up a cluster, and multiple tools that run or setup each layer.



Now let's overview the tiers that make up Antidote. The NRE Labs runtime of Antidote runs on Google Cloud Platform, so we'll use this as a example to illustrate the cluster and stack at a glance.

Infrastructure Tier	Tool Used	Explanation
Cloud Infrastructure	Terraform	All cloud infrastructure elements (VM instances, DNS entries, load balancers, etc) are defined in Terraform files. This lets us use infrastructure-as-code workflows to quickly create/delete/change infrastructure elements. Because Terraform is not a cloud DSL, if you want to host Antidote on another cloud, you only need to change the Terraform layer, made simple by Terraform Providers.
Instance Config	Ansible	Once Terraform provisions the cloud VM Linux instances, they need to be configured, such as having dependencies installed, configuration files written, processes restarted, etc. We use an Ansible playbook to initially configure all VM instances, including those that are added to the compute cluster to respond to a scaling event.
Container Scheduling	Kubernetes	Kubernetes orchestrates the services in a container cluster. When we want to provision a resource as part of a lab, we do this by interacting with Kubernetes API.
Lesson Provisioning	Syringe	Kubernetes is still fairly low-level for our purposes, and we don't want to place the burden on the lab contributor to know how Kubernetes works. So we developed a tool called Syringe that ingests a very simple lab definition, and creates the necessary resources in the cluster. It also provides an API for the web front-end (antidote-web) to know how to provision and connect to lab resources.
Web Front-End	antidote-web	We developed a custom web application based on Apache Tomcat and Apache Guacamole that serves the front end. Guacamole specifically enables an in-browser terminal experience for virtual Linux nodes or network devices in the lab topology. Jupyter notebooks handle stepping through the lab exercise steps.

The first three layers are Antidote's "infrastructure". This means we are using existing tools and software to get to a working Kubernetes cluster. The final two components make up the *Antidote Platform*, which is the bulk of the custom software developed for the Antidote project.

Antidote's infrastructure scripts provision and manages Kubernetes as opposed to using a hosted, managed solution like GKE because:

- It needs a custom CNI plugin (multus). GKE doesn't support CNI except for very tightly controlled deployments. This means that we can't have multiple interfaces per pod. We tried our best to think of alternatives but the tradeoffs were not great. Using multus is really our only option, which means the K8s deployment we're using must support CNI.
- Keeping things simple, and using only IaaS makes antidote more portable. You could deploy onto any cloud provider, or even on-premises. Initially, antidote was developed on top of GKE and in addition to the other constraints, it wasn't possible to keep GKE-specifics from leaking into the logic of the application. So, we're keeping things generic at the infrastructure layer, allowing others to pick the foundation that works for them more easily.

CHAPTER 2

Antidote Platform

2.1 Syringe - Antidote's "Brains"

2.1.1 Configuring Syringe

Syringe is configured through environment variables. Since Syringe is typically deployed on Kubernetes, the best way to pass these to Syringe is directly in the `Pod` or `Deployment` definition:

```
(...truncated...)

containers:
- name: syringe
  image: antidotelabs/syringe:latest
  imagePullPolicy: Always
  env:
  - name: SYRINGE_LESSONS
    value: "/antidote/lessons"
  - name: SYRINGE_DOMAIN
    value: "localhost"

(...truncated...)
```

If you're using antidote-selfmedicate to spin up an instance of Antidote and Syringe yourself, note that these are provided in the included Kubernetes manifest.

See the table below for a description of each, as well as information on which are required, and which have default values.

Variable Name	Required?	Default Value	Description
SYRINGE_LESSONS	Yes	N/A	Directory where the lessons are stored
SYRINGE_DOMAIN	Yes	N/A	Domain where Antidote-web is running. Used to control ingress routing
SYRINGE_GRPC_PORT	No	50099	Port to use for Syringe's GRPC service
SYRINGE_HTTP_PORT	No	8086	Port to use for the grpc-gateway REST API for Syringe
SYRINGE_TIER	No	local	Controls which lessons Syringe makes available based on lesson metadata. Can be `local`, `ptr`, or `prod`.
SYRINGE_LESSON_REPO_REMOTE	No	https://github.com/nre-learning/antidote.git	Git repo from which to clone lessons into lesson endpoint pods
SYRINGE_LESSON_REPO_BRANCH	No	master	Git branch to use in lesson endpoint pods
SYRINGE_LESSON_REPO_DIR	No	/antidote	Destination directory to use when cloning into lesson endpoint pods

2.1.2 The `syringe.yaml` File

Syringe uses a totally file-driven approach to lesson definitions. This allows us to store all lessons as "code" within a Git repository, rather than maintain a database of lesson state all the time. When syringe starts, it looks for lesson definitions within a directory, loads them into memory, and serves them directly via its API.

Note: The way that `syringe.yaml` files are put together is still a work-in-progress. Be prepared for changes to the below information, as we improve Syringe and make it (hopefully) easier to put these lesson files together.

These lesson definitions are written in YAML. A very simple example is shown below. This file describes a very simple lesson in two parts, with a single linux container for interactivity:

```
---
lessonName: Introduction to YAML
lessonID: 14
category: introductory
topologyType: none

utilities:
- name: linux1
  image: antidotelabs/utility
  sshuser: antidote
  sshpassword: antidotepassword

stages:
  1:
    description: Lists
    labguide: https://raw.githubusercontent.com/nre-learning/antidote/master/lessons/
↪lesson-14/stage1/guide.md
  2:
    description: Dictionaries (key/value pairs)
    labguide: https://raw.githubusercontent.com/nre-learning/antidote/master/lessons/
↪lesson-14/stage2/guide.md
```

(continues on next page)

(continued from previous page)

A more complicated example adds network devices to the mix. This not only adds images to the file, but we also need to add a list of connections for Syringe to place between our network devices, as well as configurations to apply to each device at each lesson stage:

```
---
lessonName: Network Unit Testing with JSNAPY
lessonID: 12
category: verification
lessondiagram: https://raw.githubusercontent.com/nre-learning/antidote/master/lessons/
↪lesson-12/lessondiagram.png
topologyType: custom

utilities:
- name: linux1
  image: antidotelabs/utility
  sshuser: antidote
  sshpassword: antidotepassword

devices:
- name: vqfx1
  image: antidotelabs/vqfxspeedy:snap1
  sshuser: root
  sshpassword: VR-netlab9
- name: vqfx2
  image: antidotelabs/vqfxspeedy:snap2
  sshuser: root
  sshpassword: VR-netlab9
- name: vqfx3
  image: antidotelabs/vqfxspeedy:snap3
  sshuser: root
  sshpassword: VR-netlab9

connections:
- a: vqfx1
  b: vqfx2
  subnet: 10.12.0.0/24
- a: vqfx2
  b: vqfx3
  subnet: 10.23.0.0/24
- a: vqfx3
  b: vqfx1
  subnet: 10.31.0.0/24
- a: vqfx1
  b: linux1
  subnet: 10.1.0.0/24

stages:
  1:
    description: No BGP config - tests fail
    labguide: https://raw.githubusercontent.com/nre-learning/antidote/master/lessons/
↪lesson-12/stage1/guide.md

    configs:
      vqfx1: stage1/configs/vqfx1.txt
      vqfx2: stage1/configs/vqfx2.txt
```

(continues on next page)

(continued from previous page)

```
      vqfx3: stage1/configs/vqfx3.txt

  2:
    description: Correct BGP config - tests pass
    labguide: https://raw.githubusercontent.com/nre-learning/antidote/master/lessons/
↪lesson-12/stage2/guide.md

    configs:
      vqfx1: stage2/configs/vqfx1.txt
      vqfx2: stage2/configs/vqfx2.txt
      vqfx3: stage2/configs/vqfx3.txt
```

2.1.3 Lab Scheduling

talk about the strange, not quote canonical way we're using k8s for lab scheduling. Syringe, multus, weave, etc.

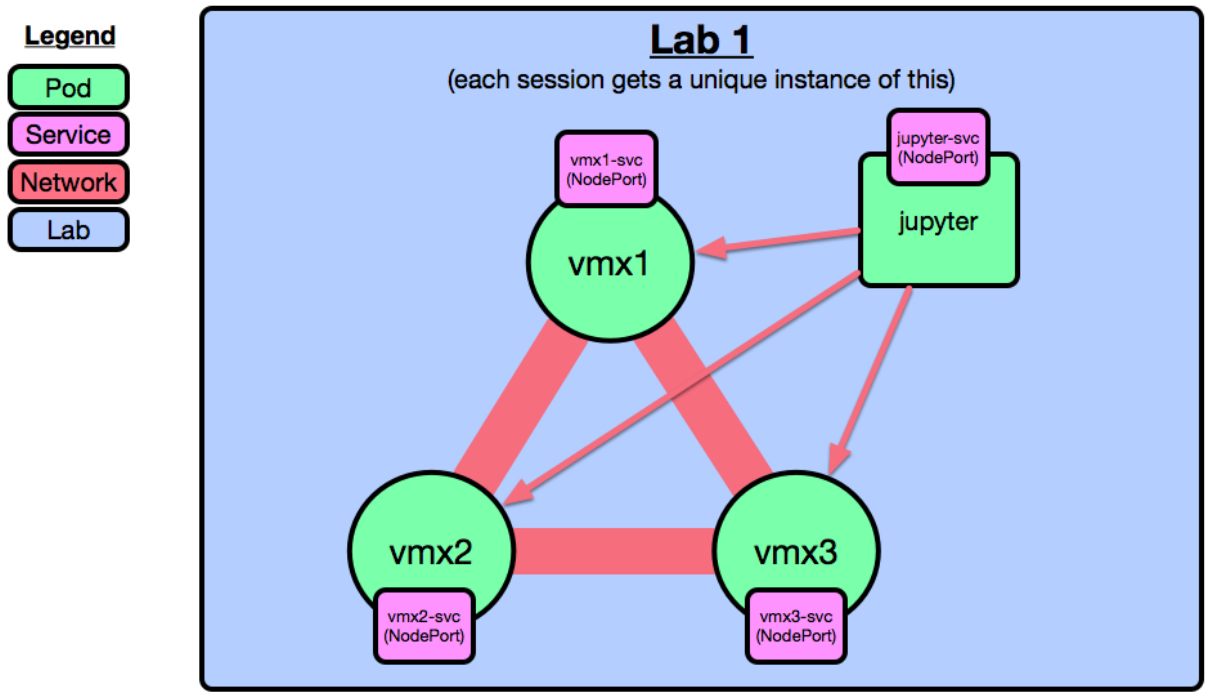
Talk about isolation

What is a Lab?

When a user logs in to the NRE Labs portal to access an exercise, a variety of provisioning activities happen automatically on the back-end. A big portion of this is defining and scheduling kubernetes resources to support that exercise.

We need Pods to run the container images for our network devices, Jupyter notebooks, and anything else required by the exercise. We need Services to make those Pods accessible to the outside world, so that the web front-end knows where to connect to those resources. Finally, we also need to define several Network objects, which is a Custom Resource Definition we define in Antidote so we can provide more granular network topologies between our Pods.

To put this under a nice umbrella term so we can more readily refer to these resources as a single cohesive unit that serves a given exercise, we're simply calling this a "lab".



Note that this is what's required for a single user to use a single lab. If multiple users are using this lab, then multiple instances of this same topology will be spun up in parallel, in isolation from each other. In addition, this is just the lab layout for this particular exercise. Other exercises will leverage a slightly different lab definition, and will likely have multiple instances of that running in a given moment as well based on usage.

NOTE - orchestrating all of these resources is done by syringe - see this page (LINK) for more details

Syringe is where the real work gets done in the Antidote project. It's responsible for taking in lesson definitions via *a YAML file*, and any configs, scripts, etc used in the lesson, and providing them to the front-end via its API.

2.2 Antidote-Web - Antidote's Front-End

TBD

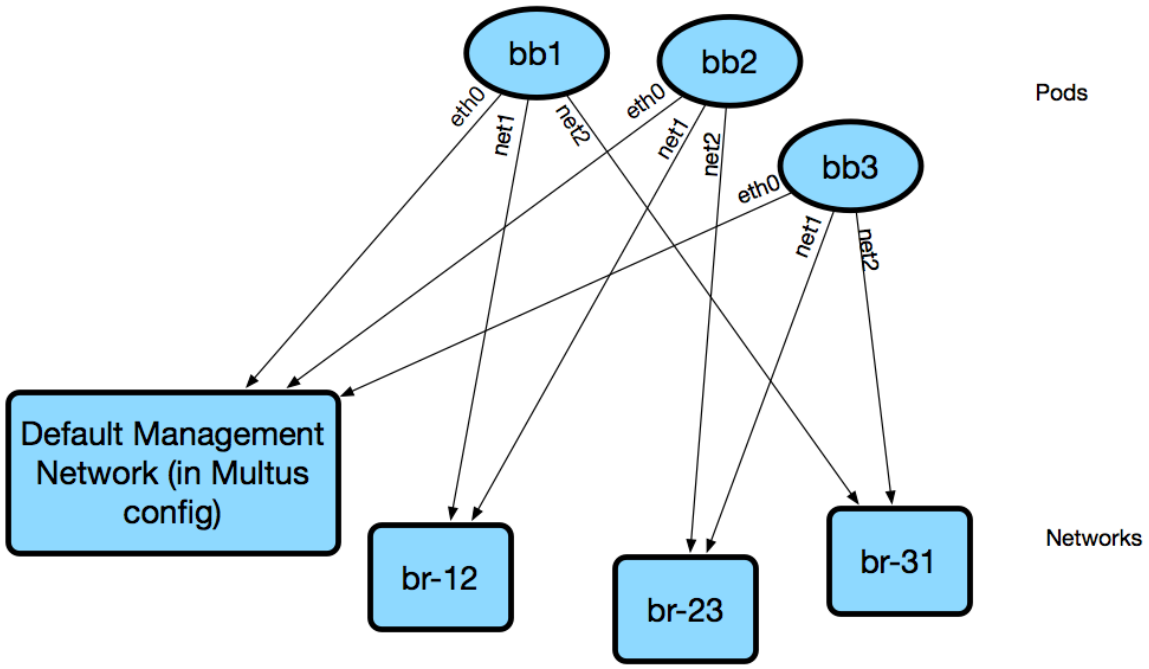
2.3 Lesson Networking

In order to connect entities within a lesson to each other, we go with a slightly unorthodox approach. Part of this design was due to constrains within the operating environment, and others are temporary solutions that will be deprecated as the solution matures. For now, here's how it works.

The TL;DR is as follows:

- Every Kubernetes pod is connected to the "main" network via its `eth0` interface. However, because we're using Multus, we can provision multiple networks for a pod. More on that in a bit.
- When we schedule lesson resources, we use affinity rules to ensure all of a lessons' resources are scheduled onto the same host.
- Depending on the resource type, and the connections described in the *lesson definition*, we may also connect additional interfaces to a pod, connected to other networks.

- Since all pods are on the same host, if we need to connect pods together directly, such as in a specified network topology, we can simply create a linux bridge and add the relevant interfaces. In the future, we will do away with affinity rules and use overlay networking instead of the simple linux bridge.



We use a default network configuration in `/etc/cni/net.d/1-multus-cni.conf` which specifies a CNI plugin that will be used by default, such as Weave. This means that all pods are configured this way for their *eth0* interface. All future networks we attach to a pod start with *net1* and so forth.

```
{
    "name": "node-cni-network",
    "type": "multus",
    "kubeconfig": "/etc/cni/net.d/multus.d/multus.kubeconfig",
    "delegates": [{
        "type": "weave-net",
        "hairpinMode": true,
        "masterplugin": true
    }]
}
```

We'll demonstrate this with a simple busybox image for simplicity.

busybox_networks.yaml:

```
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
    name: 12-net
spec:
    config: '{
        "name": "12-net",
        "type": "bridge",
        "plugin": "bridge",
        "bridge": "12-bridge",
```

(continues on next page)

(continued from previous page)

```
        "forceAddress": false,
        "hairpinMode": true,
        "delegate": {
                "hairpinMode": true
        },
        "ipam": {
            "type": "host-local",
            "subnet": "10.10.12.0/24"
        }
    }'

---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
    name: 23-net
spec:
    config: '{
        "name": "23-net",
        "type": "bridge",
        "plugin": "bridge",
        "bridge": "23-bridge",
        "forceAddress": false,
        "hairpinMode": true,
        "delegate": {
                "hairpinMode": true
        },
        "ipam": {
            "type": "host-local",
            "subnet": "10.10.23.0/24"
        }
    }'

---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
    name: 31-net
spec:
    config: '{
        "name": "31-net",
        "type": "bridge",
        "plugin": "bridge",
        "bridge": "31-bridge",
        "forceAddress": false,
        "hairpinMode": true,
        "delegate": {
                "hairpinMode": true
        },
        "ipam": {
            "type": "host-local",
            "subnet": "10.10.31.0/24"
        }
    }'
```

busybox_pods.yaml:

```
---
apiVersion: v1
kind: Pod
metadata:
name: bb1
labels:
    antidote_lab: "1"
    lab_instance: "1"
    podname: "bb1"
annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "31-net" },
            { "name": "12-net" }
    ]'
spec:
affinity:
    podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: antidote_lab
            operator: In
            values:
            - "1"
        topologyKey: kubernetes.io/hostname
containers:
- name: busybox
    image: busybox
    command:
    - sleep
    - "3600"
    ports:
    - containerPort: 22
    - containerPort: 830

---
apiVersion: v1
kind: Pod
metadata:
name: bb2
labels:
    antidote_lab: "1"
    lab_instance: "1"
    podname: "bb2"
annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "12-net" },
            { "name": "23-net" }
    ]'
spec:
affinity:
    podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: antidote_lab
            operator: In
```

(continues on next page)

(continued from previous page)

```
            values:
            - "1"
        topologyKey: kubernetes.io/hostname
containers:
- name: busybox
    image: busybox
    command:
    - sleep
    - "3600"
    ports:
    - containerPort: 22
    - containerPort: 830

---
apiVersion: v1
kind: Pod
metadata:
name: bb3
labels:
    antidote_lab: "1"
    lab_instance: "1"
    podname: "bb3"
annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "23-net" },
            { "name": "31-net" }
    ]'
spec:
affinity:
    podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: antidote_lab
            operator: In
            values:
            - "1"
        topologyKey: kubernetes.io/hostname
containers:
- name: busybox
    image: busybox
    command:
    - sleep
    - "3600"
    ports:
    - containerPort: 22
    - containerPort: 830
```

Since we set affinity rules to ensure all pods in this example run on the same host, we can see that all three pods are on node *antidote-worker-rm4m*.

```
kubectl get pods -owide
NAME      READY     STATUS    RESTARTS   AGE       IP           NODE
bb1       1/1       Running   0          6m        10.46.0.3    antidote-worker-rm4m
bb2       1/1       Running   0          6m        10.46.0.2    antidote-worker-rm4m
bb3       1/1       Running   0          6m        10.46.0.1    antidote-worker-rm4m
```

This means we can go straight to *antidote-worker-rm4m* and look directly at the linux bridges to see all of the veth pairs created for our pods connected to their respective bridges.

```
[mierdin@antidote-worker-rm4m ~]$ brctl show
bridge name bridge id               STP enabled     interfaces
12-bridge           8000.3ef2f983be58       no              veth7f22f574
                            vethb05bc7c8
23-bridge           8000.7204d78214a6       no              veth64adfee5
                            veth87397395
31-bridge           8000.5e998329ff44       no              veth4e639bb9
                            vethc8a58c24
docker0             8000.0242dc1bc14f       no
weave               8000.6e3a5b617747       no              vethwe-bridge
                            vethweeth0pl321
                            vethweeth0pl41f
                            vethweeth0plff0
```

Let's take a peek into our pods to look at the network interfaces it sees:

```
kubectl exec bb1 ip addr show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
3: net1@if46: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 0a:58:0a:0a:1f:04 brd ff:ff:ff:ff:ff:ff
    inet 10.10.31.4/24 scope global net1
    valid_lft forever preferred_lft forever
    inet6 fe80::601c:c6ff:fec6:9938/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
5: net2@if48: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 0a:58:0a:0a:0c:07 brd ff:ff:ff:ff:ff:ff
    inet 10.10.12.7/24 scope global net2
    valid_lft forever preferred_lft forever
    inet6 fe80::84bd:e3ff:fe12:59d1/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
41: eth0@if42: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1376 qdisc noqueue
    link/ether 8e:1a:a5:9f:75:ba brd ff:ff:ff:ff:ff:ff
    inet 10.46.0.3/12 brd 10.47.255.255 scope global eth0
    valid_lft forever preferred_lft forever
    inet6 fe80::8c1a:a5ff:fe9f:75ba/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
```

```
kubectl exec bb2 ip addr show

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
3: net1@if45: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 0a:58:0a:0a:0c:06 brd ff:ff:ff:ff:ff:ff
    inet 10.10.12.6/24 scope global net1
    valid_lft forever preferred_lft forever
    inet6 fe80::5c19:c5ff:fea8:e2fd/64 scope link tentative flags 08
```

(continues on next page)

(continued from previous page)

```
    valid_lft forever preferred_lft forever
5: net2@if50: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 0a:58:0a:0a:17:07 brd ff:ff:ff:ff:ff:ff
    inet 10.10.23.7/24 scope global net2
    valid_lft forever preferred_lft forever
    inet6 fe80::d8f2:58ff:fe8a:deca/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
43: eth0@if44: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1376 qdisc noqueue
    link/ether 1a:c8:5d:95:a1:ba brd ff:ff:ff:ff:ff:ff
    inet 10.46.0.2/12 brd 10.47.255.255 scope global eth0
    valid_lft forever preferred_lft forever
    inet6 fe80::18c8:5dff:fe95:a1ba/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
```

```
kubectl exec bb3 ip addr show

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
3: net1@if47: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 0a:58:0a:0a:17:06 brd ff:ff:ff:ff:ff:ff
    inet 10.10.23.6/24 scope global net1
    valid_lft forever preferred_lft forever
    inet6 fe80::dca1:79ff:fe89:a1f2/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
5: net2@if49: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 0a:58:0a:0a:1f:05 brd ff:ff:ff:ff:ff:ff
    inet 10.10.31.5/24 scope global net2
    valid_lft forever preferred_lft forever
    inet6 fe80::64e2:b0ff:fed4:952e/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
39: eth0@if40: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1376 qdisc noqueue
    link/ether 5e:e8:3f:c7:f3:a3 brd ff:ff:ff:ff:ff:ff
    inet 10.46.0.1/12 brd 10.47.255.255 scope global eth0
    valid_lft forever preferred_lft forever
    inet6 fe80::5ce8:3fff:fec7:f3a3/64 scope link tentative flags 08
    valid_lft forever preferred_lft forever
```

We can, of course, ping bb2 and bb3 from bb1 using the addresses shown above:

```
kubectl exec -it bb1 /bin/sh
/ # ping 10.10.12.6 -c3
PING 10.10.12.6 (10.10.12.6): 56 data bytes
64 bytes from 10.10.12.6: seq=0 ttl=64 time=0.101 ms
64 bytes from 10.10.12.6: seq=1 ttl=64 time=0.111 ms
64 bytes from 10.10.12.6: seq=2 ttl=64 time=0.106 ms

--- 10.10.12.6 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.101/0.106/0.111 ms
/ # ping 10.10.31.5 -c3
PING 10.10.31.5 (10.10.31.5): 56 data bytes
64 bytes from 10.10.31.5: seq=0 ttl=64 time=33.159 ms
```

(continues on next page)

(continued from previous page)

```
64 bytes from 10.10.31.5: seq=1 ttl=64 time=0.109 ms
64 bytes from 10.10.31.5: seq=2 ttl=64 time=0.103 ms

--- 10.10.31.5 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.103/11.123/33.159 ms
```

2.3.1 DNS

DNS in Antidote is provided by Kubernetes. So, if you want to reach vqfx1, simply query for *vqfx1*. You will be directed to the correponding service in your namespace. Note that each lesson + session combination gets its own namespace, which means `vqfx1` is locally significant to your lesson specifically.

CHAPTER 3

Building Antidote

3.1 Selfmedicate - Antidote's Development Environment

If you want to contribute some lessons, you'll probably want to find a way to run them locally yourself before opening a pull request. Or maybe you're looking to show some automation demos in an environment that doesn't have internet access (hello federal folks). If any of this describes you, you've come to the right place.

The antidote project is meant to run on top of Kubernetes. Not only does this provide a suitable deployment environment for the Antidote platform itself, it also allows us to easily spin up additional compute resources for running the lessons themselves. To replicate the production environment at a smaller scale, such as on a laptop, we use "minikube". This is a single-node Kubernetes cluster that is fairly well-automated. This means you can do things like run demos of Antidote lessons offline, or use this as your development environment for building new lessons.

The selfmedicate tool is a set of scripts that use minikube to deploy a full functioning Antidote deployment on your local machine.

Warning: Currently, selfmedicate only supports mac and linux. If you want to run this on Windows, we recommend executing these scripts from within a linux virtual machine, or within Windows WSL.

Warning: The `selfmedicate` script is designed to make it easy to configure a local minikube environment with everything related to Antidote installed on top. However, you'll always be well-served by becoming familiar with `minikube` itself so that you are more able to troubleshoot the environment when things go wrong. Keep a bookmark to the minikube docs handy, just in case.

3.1.1 Install and Configure Dependencies

The development environment runs within a virtual machine, so you'll need a hypervisor. We recommend Virtualbox as it is widely supported across operating systems as well as the automation we'll use to get everything spun up on top of it.

Next, you'll need minikube. This is by far the easiest way to get a working instance of Kubernetes running on your laptop. Follow the installation instructions to install, but do not start minikube.

> **Warning:** The currently supported version of minikube is v0.34.1. Please install this version specifically. You may encounter issues with other versions.

Note: The `selfmedicate` script starts a minikube VM with 8GB of RAM and 4 vCPUs by default. While this isn't a strict requirement, it's a highly advised minimum. Depending on the lessons you start, it can require quite a bit of system resources, especially if you start multiple lessons in parallel.

3.1.2 Starting the Environment

All of the scripts and kubernetes manifests for running Antidote within minikube are located in the antidote-selfmedicate repository. Clone and enter this repository:

```
git clone https://github.com/nre-learning/antidote-selfmedicate && cd antidote-
↪selfmedicate/
```

Note: By default, the selfmedicate script will look at the relative path `../antidote` for your lesson directory, and automatically share those files to the development environment. If this location doesn't exist, either place the appropriate repository there, or edit the `LESSON_DIRECTORY` variable at the top of `selfmedicate.sh`.

Within this repo, the `selfmedicate.sh` script is our one-stop shop for managing the development environment. This script interacts with minikube for us, so we don't have to. Only in rare circumstances, such as troubleshooting problems, should you have to interact directly with minikube. This script has several subcommands:

```
./selfmedicate.sh -h
Usage: selfmedicate.sh <subcommand> [options]
Subcommands:
    start    Start local instance of Antidote
    reload   Reload Antidote components
    stop     Stop local instance of Antidote
    resume   Resume stopped Antidote instance

options:
-h, --help                show brief help
```

To initially start the selfmedicate environment, use the `start` subcommand, like so:

```
./selfmedicate.sh start
```

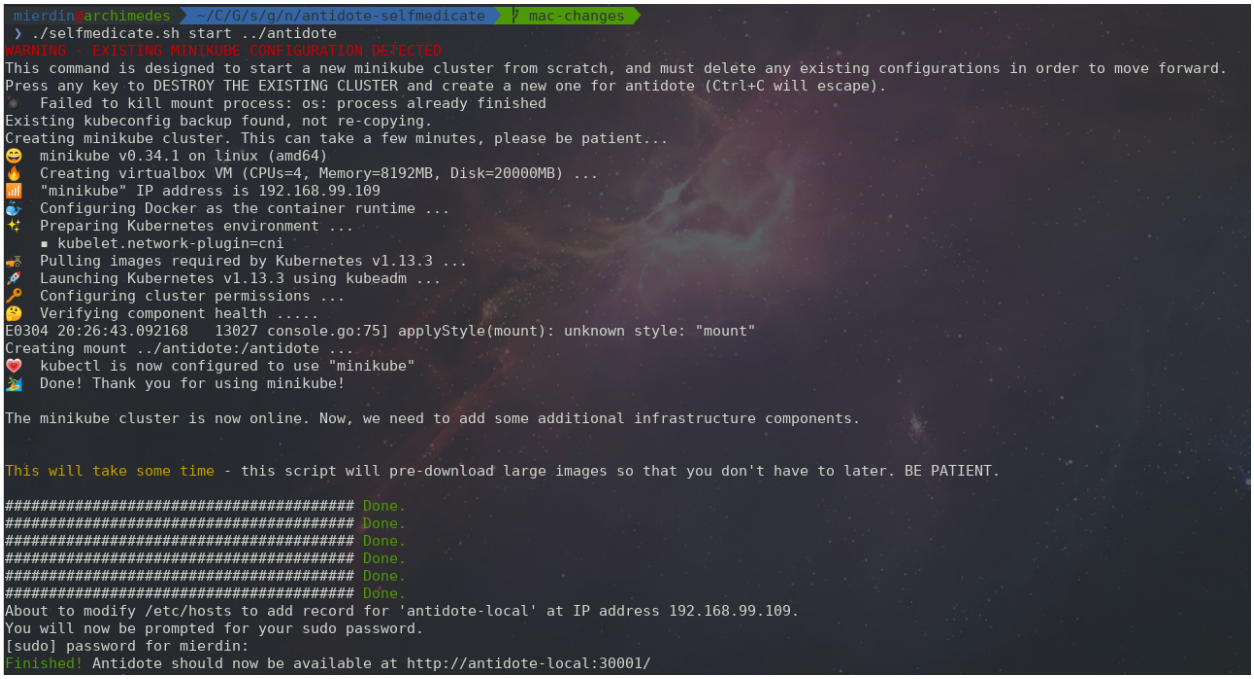
The output of this script should be fairly descriptive, but a high-level overview of the four tasks accomplished by the `selfmedicate` script in this stage is as follows:

1. `minikube` is instructed to start a Kubernetes cluster with a variety of optional arguments that are necessary to properly run the Antidote platform
2. Once a basic Kubernetes cluster is online, some additional infrastructure elements are installed. These include things like Multus and Weave, to enable the advanced networking needed by lessons.
3. Platform elements like `syringe` and `antidote-web` are installed onto the minikube instance.

4. Common and large images, like the `vqfx` and `utility` images are pre-emptively downloaded to the minikube instance, so that you don't have to wait for these to download when you try to spin up a lesson.
5. Once all the above is done, the script will ask for your sudo password so it can automatically add an entry to `/etc/hosts` for you. Once this is done, you should be able to access the environment at the URL shown.

> **Warning:** Each of these steps are performed in sequence, and will wait for everything to finish before moving on to the next step. This script is designed to do as much work as possible up-front, so that your development experience can be as positive as possible. As a result, the first time you run this command can take some time. BE PATIENT. Also note that if you destroy your minikube instance, you'll need to redo all of the above. If you want to just temporarily pause your environment, see the section below on the `stop` and `resume` subcommands.

The below screenshot shows this command in action, for your reference. You should see more or less the same thing on your environment.



Once this is done, the environment should be ready to access at the URL shown by the script.

3.1.3 Iterating on Lessons

One of the biggest use cases for running `selfmedicate` is to provide a local instance of the antidote platform for building and testing curriculum contributions. As was briefly mentioned in the `start` section above, the `selfmedicate` script takes care of mapping the files on your local filesystem into minikube and again into the Syringe pod to ensure it sees the lessons you're working on.

This means you can work on lessons all on your local machine without having to bother editing environment variables or committing your content to a Git repository.

Once you have a working antidote installation according to the previous section, you'll notice that the web portal shows the lessons as they existed when you initially started the platform. If you want to apply any changes you've made locally, you need to run the `reload` subcommand of the `selfmedicate` script:

```
./selfmedicate.sh reload
```

This command will take care of restarting Syringe, so that it can reload the content you've changed on your filesystem.

3.1.4 Pausing and Resuming Environment

As mentioned above, if you destroy the minikube environment, you'll need to perform the `start` command all over again. However, it would be nice to be able to stop the environment temporarily, and resume later without installing everything over again from scratch.

Fortunately, the `stop` and `resume` subcommands do just this for us. To stop/pause the environment, run:

```
./selfmedicate.sh stop
```

To resume, run:

```
./selfmedicate.sh resume
```

The `resume` command is important to run, since this re-executes minikube with the optional arguments needed by Antidote, so make sure to use this, rather than trying to use `minikube start` directly.

3.1.5 Troubleshooting Self-Medicate

The vast majority of all setup activities are performed by the `selfmedicate` script. The idea is that this script shoulders the burden of downloading all the appropriate software and building is so that you can quickly get to focusing on lesson content.

However, issues can still happen. This section is meant to direct you towards the right next steps should something go wrong and you need to intervene directly.

Note: If your issue isn't covered below, please open an issue on the selfmedicate repository.

Cannot connect to the Web Front-End

It's likely that the pods for running the Antidote platform aren't running yet. Try getting the current pods:

```
~$ kubectl get pods
NAME                                        READY   STATUS    RESTARTS   AGE
antidote-web-99c6b9d8d-pj55w                2/2     Running   0          12d
nginx-ingress-controller-694479667b-v64sm   1/1     Running   0          12d
syringe-fbc65bdf5-zf4l4                     1/1     Running   4          12d
```

You should see something similar to the above. The exact pod names will be different, but you should see the same numbers under the `READY` column, and all entries under the `STATUS` column should read `Running` as above.

In some cases the `STATUS` column may read `ContainerCreating`. In this case, it's likely that the images for each pod is still being downloaded to your machine. You can verify this by "describing" the pod that's not `Ready` yet:

```
kubectl describe pods -n=kube-system kube-multus-ds-amd64-ddxqc
Name:               kube-multus-ds-amd64-ddxqc
....truncated....
Events:
```

(continues on next page)

(continued from previous page)

```
Type    Reason     Age   From               Message
----    ------     ----  ----               -------
Normal  Scheduled  19s   default-scheduler  Successfully assigned kube-system/kube-
↪multus-ds-amd64-ddxqc to minikube
Normal  Pulling    17s   kubelet, minikube  pulling image "nfvpe/multus:latest"
```

In this example, we're still waiting for the image to download - the most recent event indicates that the image is being pulled. The `selfmedicate.sh` script has some built-in logic to wait for these downloads to finish before moving to the next step, but in case that doesn't work, this can help you understand what's going on behind the scenes.

If you're seeing something else, it's likely that something is truly broken, and you likely won't be able to get the environment working without some kind of intervention. Please open an issue on the antidote-selfmedicate repository with a full description of what you're seeing.

Lesson Times Out While Loading

Let's say you've managed to get into the web front-end, and you're able to navigate to a lesson, but the lesson just hangs forever at the loading screen. Eventually you'll see some kind of error message that indicates the lesson timed out while trying to start.

This can have a number of causes, but one of the most common is that the images used in a lesson failed to download within the configured timeout window. This isn't totally uncommon, since the images tend to be fairly large, and on some internet connections, this can take some time.

There are a few things you can try. For instance, `kubectl describe pods <pod name>`, as used in the previous section, can tell you if a given pod is still downloading an image.

We can also use the `minikube ssh` command to send commands into the minikube VM and see the results. For instance, to check the list of docker images that have been successfully pulled:

Note: minikube ssh docker image list

This is the same as running `docker image list`, but it's done from inside the minikube VM for you. Similarly, if you wanted to manually pull an image ahead of time, you could run `minikube ssh docker image pull <image>`.

Note: The `selfmedicate` script downloads the most common images in advance to try to reduce the likelihood of this issue, and to generally improve the responsiveness of the local environment. However, it can't do this for all possible images you might want to use. If you know you'll use a particular image commonly, consider adding it to the `selfmedicate` script, or manually pulling it within the minikube environment ahead of time.

3.1.6 Validating Lesson Content

Syringe, the back-end orchestrator of the Antidote platform, comes with a command-line utility called `syrctl`. One of the things `syrctl` can do for us is validate lesson content to make sure it has all of the basics to work properly. To run this command, you can compile the syrctl binary yourself from the Syringe repo, or you can execute the pre-built docker container:

```
docker run -v .:/antidote antidotelabs/syringe syrctl validate
```

In the very near future, `syrctl` will be added to the CI pipeline for Antidote so that you know in your PR if there's any issues.

3.2 Deploying Antidote into Production

This is NOT the guide for running a local instance of Antidote. That can be found in the *build local instructions*. These instructions are if you wish to leverage the Terraform and Ansible scripts to get a production-quality instance of Antidote running in the cloud.

3.2.1 Prerequisites

- Git
- Terraform
- Ansible

3.2.2 Infrastructure Setup

Follow the instructions for installing the Google Cloud SDK, and then run these commands to authenticate:

```
gcloud auth login
gcloud auth application-default login
gcloud config set compute/zone us-west1-b
gcloud config set compute/region us-west1
```

You will also want to set up a billing account, and write down the billing account ID (you can find this on the [billing console](https://console.cloud.google.com/billing)). You'll need that when we run terraform.

Make sure to clone this repo so you can get access to the Terraform configs and Ansible playbooks for provisioning the environment:

```
git clone https://github.com/nre-learning/antidote && cd antidote
```

All of Antidotes infrastructure is defined in Terraform configurations. So we just need to run *terraform apply*:

Note: At this point, Terraform will prompt you to provide the billing account ID you just retrieved in the previous step. It goes without saying but **NOTE THAT THIS WILL MAKE CHANGES IN YOUR ENVIRONMENT**. Review the output of *terraform apply* or *terraform plan* to determine the scope of the changes you're making:

```
cd infrastructure/
terraform apply
```

Once terraform creates all of the necessary resources, we first need to do a little manual work. We need to create an A record in the GCE dashboard for *vip.labs.networkreliability.engineering*, and add all of the provisioned NAT IPs (external) for each of the instances in the *workers* instance group. Do this before moving on. Eventually this will be replaced with a proper load balancing setup.

Next, bootstrap the cluster:

```
./bootstrapcluster.sh
```

3.2.3 Platform Setup

Next we need to start the services that will power Antidote. At this point, you should now be able to run kubectl commands. Verify with:

```
kubectl get nodes
```

Enter the *platform* directory and execute the shell script to upload all of the platform kubernetes manifests:

```
cd ../platform/
./bootstrapplatform.sh
```

Note: may need to restart the coredns pods

3.2.4 Cleaning Up

As expected, clean up with *terraform destroy*

Please use the menu above to get to the relevant instructions for how you're trying to build Antidote.

CHAPTER 4

Contributing to Antidote

4.1 Contributing to the Curriculum

Thanks for contributing to the Antidote curriculum. Before starting your work, please read this document in its entirety, so your work has the maximum impact and your time is best spent.

4.1.1 Step 1 - Ask Around!

Antidote is still in tech preview, so there is a lot of work going on all the time. Before you get started, make sure you don't conflict with any existing or current work!

The first step is to peruse the existing curriculum issues. If someone's already working on something related to the curriculum, there's a good chance that there will be an issue there. If not, please first open an issue so the community can have a chance to provide feedback on your idea before you spend time building it. There's a chance that something already exists, or maybe someone else is planning to work on it, and evangelizing your idea first gives you an opportunity to combine forces with these existing efforts if they exist.

Once you feel like you've gotten good feedback, and you have a good plan in place, read the following section for some guidelines on how to write a really awesome lesson.

4.1.2 Step 2 - Plan It Out

Lessons in NRE Labs should demonstrate something useful to a network engineer. For instance, the first lesson is a simple NAPALM based python tool that collects system information about a target device.

If a user can easily replicate what is shown in the lab so that it can help them in production, then this would be even better.

There are multiple ways to build lessons in Antidote.

One-off

The simplest method is to have single one-off lessons that do not have any direct relationship to other lessons.

Repeat

Some lessons can be repeated 2 or 3 times. For instance, in addition to the NAPALM lesson, you could show the user how to collect system information using an alternate method. You should explain why a network engineer would want to choose one method over another. In the case of the first lesson, NAPALM is a somewhat limited tool. If the user needs additional information, they would need to do something different. They could use PyEZ, for instance.

Workflow

Some lessons could be a group of inter-related tasks. A troubleshooting workflow that helps a network engineer locate a device in the network, or the path between two devices, could be broken up into a set of distinct tasks. Not every task has to be automated, but some could be, and the lessons could reflect this.

Considerations

There are a number of languages, tools, and libraries/packages that could be leveraged to build a lesson. Consider using open-source tools for the lessons, or tools that are at least free. This helps ensure that a user could more easily replicate what is shown in the lesson.

Note: For security reasons, internet access will not be enabled for labs hosted within *NRE Labs*. All lessons should be totally self-contained and not rely on external resources to properly function when the lesson is being run.

4.1.3 Step 3 - Put It Together

Okay. You've determined that you have a good idea for a lesson, and no one else is working on it. Let's get you started!

The best ally to have when buliding lessons is a local instance of Antidote running on your own laptop. Ironing out all of the bugs locally before you submit a pull request makes the review process much smoother. For more info, see the *local build* documentation.

Once you're ready to start building a lesson, you'll need to create a *syringe.yaml file*. This is the primary metadata file for a lesson, and it's here where we declare the attributes (such as the name of the lesson and what course it belongs to), what resources the lesson needs, and how they connect to each other (if applicable).

Note: It's very important to get the `syringe.yaml` file right, because it's so central to the definition of a lesson. Please refer to the *syringe.yaml docs* for detailed documentation on this file.

This is really the only "required" file for a lesson, but in order to be useful, the `syringe.yaml` file will need to refer to other files like configs, markdown files, and more, in order to be useful. Take a look at the existing lessons and see how those lessons are laid out. What you'll need in your lesson directory will vary wildly, depending on the contents of your `syringe.yaml` file.

4.1.4 Step 4 - Get It Reviewed

Once you've got your changes together, and pushed to a branch of your own (i.e. in a fork of the Antidote repo), open a pull request.

Here are a few things that reviewers should be on the lookout for when reviewing new contributions to the curriculum, either for new or existing lessons. If you're contributing to the curriculum, you should be aware of these guidelines, to make the review process much smoother.

- Can a user get through a lesson stage quickly? Are we letting them get to a quick win as soon as practical while still teaching quality content?

- Does the new or changed lesson adhere to the spirit of Antidote lessons laid out in this document?
- For new lessons, does the lesson guide (or jupyter notebook if applicable) look nice? Does the author attribute themselves?
- Is the lesson guide(s) easy to follow?
- Are any documentation updates also needed?
- Is the CHANGELOG updated properly?
- Can we show this in NRE labs? Usage rights?
- Does this follow the *Lesson Image Requirements*?
- Is the business benefit clear from this lesson? How easy is it for people to link this content with their day-to-day?

4.1.5 Appendix - Lesson Contribution FAQ

NAPALM Can't Find My Configs.

This is likely due to the way you've deployed syringe.

In the selfmedicate repo, there are a number of kubernetes manifests useful for running antidote locally. However, there are some defaults here you'll likely want to change. In particular, if you're making lesson changes in a branch or fork (which is ideal if you want to open a PR) you will want to make sure you update the syringe deployment in two places:

- The init-container definition, where the `antidote` repo is cloned into the syringe pod
- Syringe's `SYRINGE_LESSON_REPO_REMOTE` and `SYRINGE_LESSON_REPO_BRANCH` environment variables.

Be sure to re-deploy syringe using `kubectl apply -f syringe.yaml` once you've made the appropriate changes. If you've already made these changes and it still doesn't work, make sure syringe is using the latest copy of your repo by deleting the syringe pod. The Syringe deployment will re-deploy a new pod with a freshly-cloned version of your lesson repo.

4.2 Contributing to the Antidote Platform

If you want to propose a change to Antidote, Syringe, Antidote-web, or any other member project, contributions are welcome! Just be patient, as these components are still not considered quite stable, so you should reach out to the maintainers before getting too deep into the weeds on a change, so you don't waste your time.

4.3 Antidote Release Process

The Antidote project is composed of three primary Github repositories:

- Antidote - contains infrastructure and platform configurations, as well as the codified lesson definitons.
- Syringe - contains code for the back-end orchestrator between Kubernetes and the front-end
- Antidote-web - contains code for the front-end web application.

Each of these repositories maintain semantic versioning whenever a new release comes out, and they're released in sync with each other. For instance, when 0.1.3 came out, this version was created for each of these projects, and each project is intended to be deployed together at the same version.

There's no pre-determined schedule for releases. Generally we try to do a release every two weeks or so, but this is a guideline. This could vary based on conflicts with holidays, or based on incoming changes to any of the projects in Antidote.

The antidote-ops repository is a StackStorm pack for managing Antidote in production, including workflows for creating release branches/tags in Git, uploading tagged Docker images to Dockerhub, and deploying versions of the software to Kubernetes.

Whenever a new release is created, the CHANGELOG notes from each project is summarized underneath the corresponding Antidote release notes for that release. Stay tuned to that link as well as the NRE blog for updates there.

Once a release is created, it's first deployed to the *ptr* . This is a sibling site to the main production instance where new features and content can be tested before making it live to the production site. Generally, releases are deployed to the PTR as soon as they're available. They'll be tested there for at least a few days, and as long as there aren't any serious issues, the live site will also be updated to this version a few days later.

4.3.1 NRE Labs' "Public Test Realm"

The "production" runtime of NRE Labs is located at https://labs.networkreliability.engineering/. It is considered to be mostly stable (though still tech preview for now). It's deployed using tagged versions of git repositories and docker images, and goes through a decent amount of regular testing, as well as far fewer changes than the underlying `antidote` software projects.

However, there's a version of NRE Labs that is deployed every time a change to *master* is detected on one of Antidote's repositories, called the "public test realm". It's located at: https://ptr.labs.networkreliability.engineering/.

The NRE Labs PTR will be quite similar to the "production" copy, but there are a few important differences to discuss:

- Everything is deployed via `master` or the current release branch, and is inherently unstable. We will try to keep things working, but it **is** a test realm. There be dragons here.
- All lessons in the `antidote` repository are shown, despite the *disabled* property in the syringe file.
- A black bar with the latest commit ID for all three projects (`antidote`, `antidote-web`, `syringe`) is shown at the bottom of each page.

The goal with the PTR is to make it easier to test release candidates for Antidote. Once the PTR has the changes we want in the next release, and we've put it through some testing, we can more confidently release the next version. It will also help contributors see their changes more quickly, rather than waiting for a release. Finally, it also gives users a chance to see upcoming lessons, and provide early feedback.

4.4 Lesson Image Requirements

Because Antidote runs on Kubernetes, all lessons are powered by Docker images. However, there are a few rules for including images in Antidote.

- All files necessary for building a lesson image must be included in the lesson directory. Lessons hosted on NRE Labs must use images built via our tooling. No external images will be used.
- Any images meant to be used as type `device` or `utility` must be configured to listen for SSH connections on port 22 and support the credentials antidote/antidotepassword.

In general, there are two area in which to contribute to Antidote:

- The Lesson Curriculum - this is the most popular place to contribute to Antidote. Here, you can change or add new lessons that describe automation workflows. Before getting started, please read the *lesson contribution guide*.

- The Antidote Platform - there are a number of software projects that power Antidote, and these are also open source. Please visit the *platform contribution guide* before looking to contribute here.

Before starting work on a contribution, please read the relevant contribution guide; they contain vital information to ensure your time is well-spent.

CHAPTER 5

Roadmap

This is a new fast-moving project in alpha and under much development still.

The Antidote roadmap will move faster than its reference runtime *NRE Labs*. And the Antidote infrastructure is eventually expected to move slower than the contributions and releases of new Lessons.

If you find an issue to report, or want to work on Antidote infrastructure or curriculum, please use the project issues list on GitHub.

CHAPTER 6

NRE Labs

NRE Labs is the flagship and reference deployment of Antidote.

While NRE Labs aims to be intuitive to use without documentation, lab contributors and Antidote developers should familiarize themselves with concepts of the learning curriculum and in the future, gamification.

6.1 Taxonomy

In the NRE Labs curriculum there are several courses. A course is a theme and grouping of many related lessons. A lesson is a common task or workflow of many small steps. A lab is one small step or several that can be explained easily so that a user can understand and accomplish the labs in a matter of minutes.

Here is the hierarchy overview:

- Curriculum: the whole thing (NRE Labs)
- Course: Like a category or topic of study
- Lessons: A set of exercises that make sense together to teach something with a common set of resources. Usually a NetOps task.
- Labs (also known as stages): A set of related steps forming a workflow in a Lesson

6.2 Lesson and Lab Relationship

NRE Labs is supposed to be fast, easy and fun to use. Labs should not take long to complete, so a Lesson is usually made up of a handful or more Labs.

Lessons are designed to solve a problem various ways to maximize learning. It is helpful to think of solving the same workflow different ways with different kinds of Labs. That way in one Lesson or Course we teach several ways of doing something, and different skills. We can reason about automating workflows into a Lab in the following ways:

- On-box and Off-box
- Single or Multi step
- Single or Multi node
- Going direct-to-box or going through an higher-level controller or collector
- Sequential/Serial/Synchronous or In-Parallel/Asynchronous (like map-reduce)

6.3 Courses

While in the initial release of NRE Labs, we'll only have a few Courses, eventually we'll have more. Recall that a Course is a theme or grouping of many related Lessons.

Here is a list of Courses currently under consideration and development:

1. Troubleshooting and Common Tasks
2. Configuration
3. Telemetry
4. Security
5. Testing and Validation
6. Infrastructure as code
7. Event-driven infrastructure

6.4 NRE Labs Roadmap

The curriculum roadmap for NRE Labs will evolve separately from the roadmap and releases of the Antidote infrastructure. In general as infrastructure matures, the curriculum will release more frequently with new expansions.

If you want to propose a Lesson or build one yourself, please refer to the project's contributing guide and search the issues before you create a new enhancement- and curriculum-labelled issue. If you know which course it matches, then try to find the appropriate label for it, or indicate it in the description. The team welcomes and values your suggestions and contributions.

CHAPTER 7

Community Resources

We're truly interested in building a community around both the Antidote platform, as well as the NRE Labs curriculum. To that end, we've created several diverse resources for getting involved with the project.

7.1 Github

Everything we produce in the Antidote project is on Github underneath the nre-learning organization. Underneath that organization, you'll find all kinds of repositories, including the platform software, operational tooling, blogs, and more.

While there is no single repository that hosts everything, the canonically "main" repository is probably the Antidote repo. Most subcomponents of the Antidote project are hosted in their own dedicated repositories, but this will be a good starting point, as the README does contain references to other repositories you may also want to know about.

Finally, all project planning related to the Antidote platform, such as for subcomponents like `syringe` and `antidote-web` are done in Github projects. Generally while we're working on a release, we'll open up a new project for that release, and all current status for that release can be found within that project. So, if you're looking for a status on current development efforts, this is where you want to go.

7.2 Twitter

Follow @NRELabs on twitter for periodic updates on NRE Labs and the underlying platform.

7.3 Discord Channel

We use Discord for chat and audio/video collaboration. You can think of it like Slack, but even better, because it's more purpose-built for some of things we want to do in the future with NRE Labs, and the authentication story is a bit better. :)

Join our Discord channel via this link.

7.4 Mailing List

For those that prefer email for communication, we have two mailing lists:

- Developers
- Users

7.5 Weekly Livestreams and Other Videos

We run a weekly livestream on Twitch at 10AM Pacific Time. Right now this is fairly informal - some weeks we might dive deep into a topic, other weeks we might just casually chat about what's next on the project plan.

Check out our Twitch channel to watch live, and interact with us in the chat. Ask questions or provide feedback to us live, and we might talk about it on stream!

If you can't join live, check out our YouTube channel, where we post the recordings for all our streams, as well as all other video content.

7.6 Blog

The Network Reliability Engineering blog is where we like to write about NRE concepts and any big changes to NRE Labs. You may also be interested to know that this blog is powered by Hugo, and maintained in Github, so if you want to write a guest post by opening a pull request to that repo, we'd love to have you!

7.7 Trello Board

For slightly less technical project planning around the Antidote community, see our Trello Board. This is where we plan anything that doesn't fit neatly into a platform release plan, which we manage in Github projects.

Antidote is a community initiatve to democratize interactive, dependency-free learning. With Antidote, you can create compelling, interactive learning experiences by building turnkey lesson materials that are fully browser-based, placing minimal burden on the learner to come with pre-requisite knowledge or environment configurations.

Antidote is composed of a set of is an open-source project aimed at making automated network operations more accessible with fast, easy and fun learning. It teaches from-scratch network automation skills for network reliability engineers (NREs) and other NetOps pros or amateurs.

Antidote isn't just one application, but rather, a set of smaller applications that work together to provide this learning experience. The antidote-selfmedicate repository is the simplest way to get this going, as it's, designed to allow you to run everything on your laptop, using *minikube*. This is very useful if you're looking to develop some lessons and need an easy way to test them out without a lot of setup. See the *build local* instructions for more info on that.

In case you're looking to run Antidote in more of a public-facing, production capacity, the main Antidote repository contains terraform configurations, kubernetes manifests, and scripts necessary for running all of Antidote's components in the cloud. More information for spinning this up can be found in the *production* guide.

In fact, the reference runtime and use case for Antidote is *NRE Labs* and Antidote is the project behind it. NRE Labs is a free site and training service on the web sponsored by Juniper Networks, but anyone interested can run their own copy of Antidote.

Often the first step to learning about network automation is the hardest: you need to setup complex virtual environments, labs, or worse risk experimenting in production. NRE Labs was built to teach skills right in your web browser and let you deal with real tools, code and network devices.

What's more is that often at the outset of the network automation journey, when one thinks about what to automate, the answer is "the network!" But specifics about workflows are prerequisite to automating them. That's why the community here has created lessons and labs with real-life NetOps workflows. And everything you see and use is applicable and open for you to use back in your own environments.

Contributions are welcome.

Please note that NRE Labs is currently in limited tech preview.

Join the community on our Slack space and channel #nre_labs