Pivotal

Course developed by
Pivotal Academy

Pivotal Cloud Foundry Developer v1.7

APPLICATION SECURITY GROUPS AND LOG DRAIN

Application Security Groups

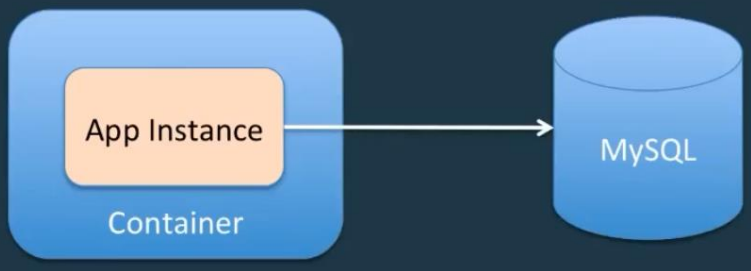
Setting up Application Security Groups is generally done by the admin, but as a developer you have tools to reason about the platform and how it is running your applications.

Agenda

1. Application Security Groups Overview

Application Security Groups

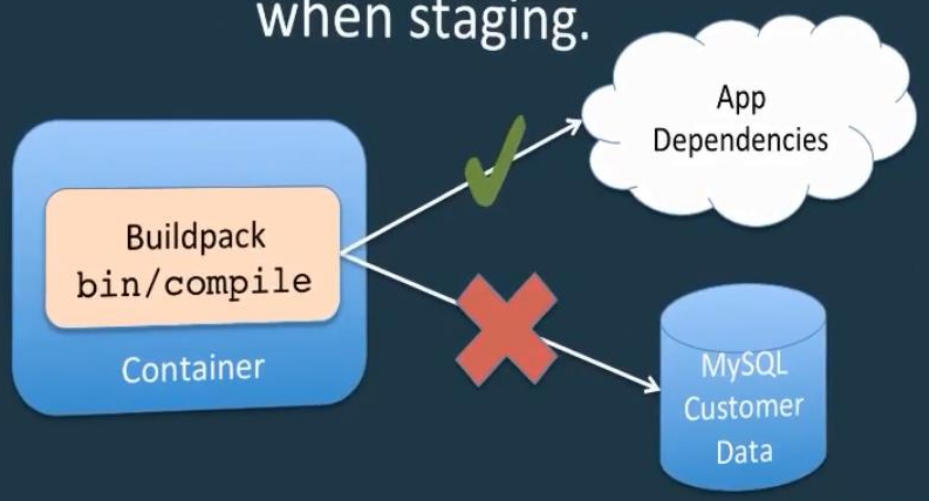
Are virtual firewalls that control egress/ outbound traffic for applications.



https://docs.pivotal.io/pivotalcf/adminguide/app-sec-groups.html

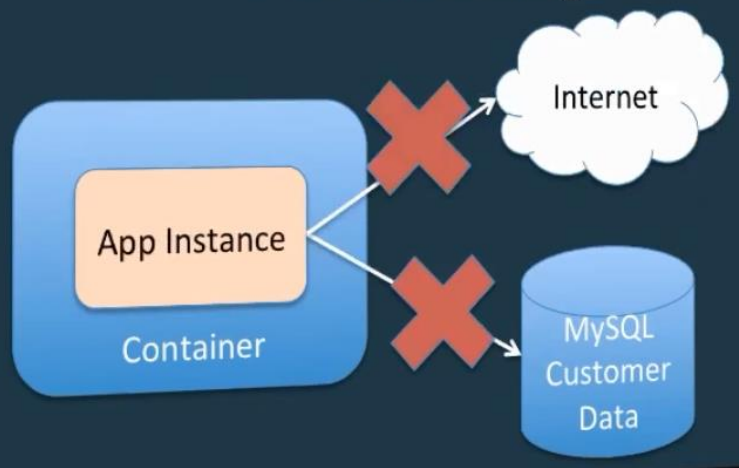
Staging Security Groups

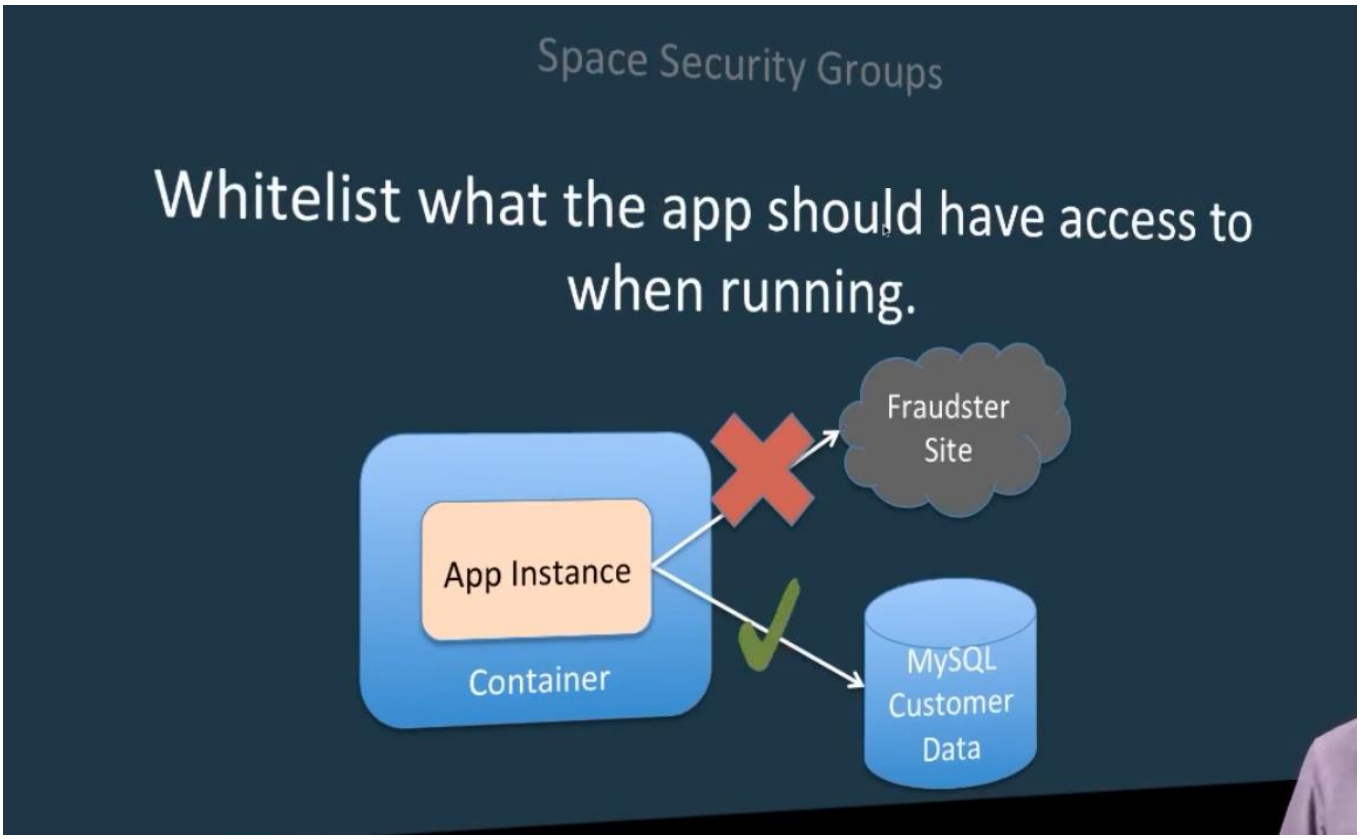
Whitelist what the app should have access to when staging.



Running Security Groups

Whitelist what all apps should have access to when running.





You can then apply this to your CF environment

```
DROBERTS-MBPRO:attendee-service droberts$ cf t

API endpoint:   https://api.run.haas-39.pez.pivotal.io (API version: 2.54.0)
User:           droberts@pivotal.io
Org:            dave
Space:          dev
DROBERTS-MBPRO:attendee-service droberts$
```

We can see that we are currently logged in as a developer and not an admin.

```
DROBERTS-MBPRO:attendee-service droberts$ cf security-groups
Getting security groups as droberts@pivotal.io
OK

     Name       Organization   Space
#0   all_open
DROBERTS-MBPRO:attendee-service droberts$
```

We can use the ***$cf security-groups*** command to see all the SGs currently running and are bound to our current Org and Space. We currently have 1 SG called **all_open**.

```
DROBERTS-MBPRO:attendee-service droberts$ cf security-groups
Getting security groups as droberts@pivotal.io
OK

     Name       Organization   Space
#0   all_open
DROBERTS-MBPRO:attendee-service droberts$ cf security-group all_open
Getting info for security group all_open as droberts@pivotal.io
OK

Name    all_open
Rules
        [
                {
                        "destination": "0.0.0.0-255.255.255.255",
                        "protocol": "all"
                }
        ]

No spaces assigned
DROBERTS-MBPRO:attendee-service droberts$
```

We can investigate the details of an Application SG further using the ***$ cf security-group all_open*** command as above, this shows that the SG opens all destinations and IP addresses and all protocols.

```
DROBERTS-MBPRO:attendee-service droberts$ cf login
API endpoint: https://api.run.haas-39.pez.pivotal.io

Email> admin

Password>
Authenticating...
OK

Select an org (or press enter to skip):
1. system
2. dave
3. sg-org

Org> 2
Targeted org dave

Targeted space dev



API endpoint:   https://api.run.haas-39.pez.pivotal.io (API version: 2.54.0)
User:           admin
Org:            dave
Space:          dev
DROBERTS-MBPRO:attendee-service droberts$
```

Let us now log in as an admin and again target the same Org and Space as earlier to see what checking the SGs look like.

```
DROBERTS-MBPRO:attendee-service droberts$ cf security-groups
Getting security groups as admin
OK

     Name          Organization   Space
#0   all_open
#1   metrics-api   system         metrics
DROBERTS-MBPRO:attendee-service droberts$
```

As an admin, using the ***$ cf security-groups*** command, we now get a list of all the ASGs available within our PCF account.

```
DROBERTS-MBPRO:attendee-service droberts$ cf staging-security-groups
Acquiring staging security group as admin
OK

all_open
DROBERTS-MBPRO:attendee-service droberts$
```

The ***$ cf staging-security-groups*** command will tell us all what SGs are active in the staging environment when we are creating the droplets. The all_open SG will allow us to be able to pull down all the dependencies that our application needs during staging.

```
DROBERTS-MBPRO:attendee-service droberts$ cf staging-security-groups
Acquiring staging security group as admin
OK

all_open
DROBERTS-MBPRO:attendee-service droberts$ cf running-security-groups
Acquiring running security groups as 'admin'
OK

all_open
DROBERTS-MBPRO:attendee-service droberts$
```

We can use the ***$ cf running-security-groups*** command to see all the system-wide SGs as above. Both the staging-security-groups and the running-security-groups are system-wide SGs that apply across all Orgs and Spaces that can be targeted. A single SG can also be used within multiple Orgs and Spaces.

```
DROBERTS-MBPRO:attendee-service droberts$ cf unbind-running-security-group all_open
Unbinding security group all_open from defaults for running as admin
OK



TIP: Changes will not apply to existing running applications until they are restarted.
DROBERTS-MBPRO:attendee-service droberts$
```

We can lock down all egresses into our application by using the ***$ cf unbind-running-security-group all_open*** command above. Next, let us see how this unbinding the all_open SG will affect our app.

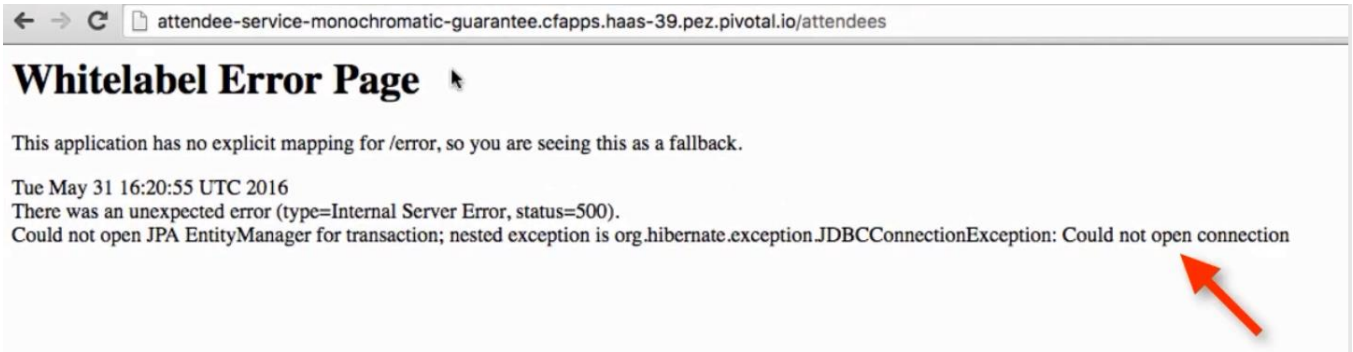
```
DROBERTS-MBPRO:attendee-service droberts$ cf restart attendee-service
Stopping app attendee-service in org dave / space dev as admin...
OK

Starting app attendee-service in org dave / space dev as admin...

0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
1 of 1 instances running

App started


OK

App attendee-service was started using this command `CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_jre/bin/java-
calculator-2.0.1_RELEASE -memorySizes=metaspace:64m.. -memoryWeights=heap:75,metaspace:10,native:10,stack:5 -memoryI
,metaspace:100% -totMemory=$MEMORY_LIMIT) && JAVA_OPTS="-Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PWD/.java-b
_jre/bin/killjava.sh $CALCULATED_MEMORY" && SERVER_PORT=$PORT eval exec $PWD/.java-buildpa  idk_jre/bin/java $
D/.:$PWD/.java-buildpack/spring_auto_reconfiguration/spring_auto_reconfiguration-1.10.0_RE  org.springframew
arLauncher`

Showing health and status for app attendee-service in org dave / space dev as admin...
```

```
App started


OK

App attendee-service was started using this command `CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_jre/bin/java-
calculator-2.0.1_RELEASE -memorySizes=metaspace:64m.. -memoryWeights=heap:75,metaspace:10,native:10,stack:5 -memoryI
,metaspace:100% -totMemory=$MEMORY_LIMIT) && JAVA_OPTS="-Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PWD/.java-b
_jre/bin/killjava.sh $CALCULATED_MEMORY" && SERVER_PORT=$PORT eval exec $PWD/.java-buildpack/open_jdk_jre/bin/java $
D/.:$PWD/.java-buildpack/spring_auto_reconfiguration/spring_auto_reconfiguration-1.10.0_RELEASE.jar org.springframew
arLauncher`

Showing health and status for app attendee-service in org dave / space dev as admin...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: attendee-service-monochromatic-guarantee.cfapps.haas-39.pez.pivotal.io
last uploaded: Fri May 27 21:36:43 UTC 2016
stack: unknown
buildpack: java-buildpack=v3.6-offline-https://github.com/cloudfoundry/java-buildpack  in open-jdk
1 open-jdk-like-memory-calculator=2.0.1_RELEASE spring-auto-reconfiguration=1.10.0_

     state     since                    cpu    memory           disk           de
#0   running   2016-05-31 11:19:58 AM   0.1%   449.5M of 512M   152.4M of 1G
DROBERTS-MBPRO:attendee-service droberts$
```

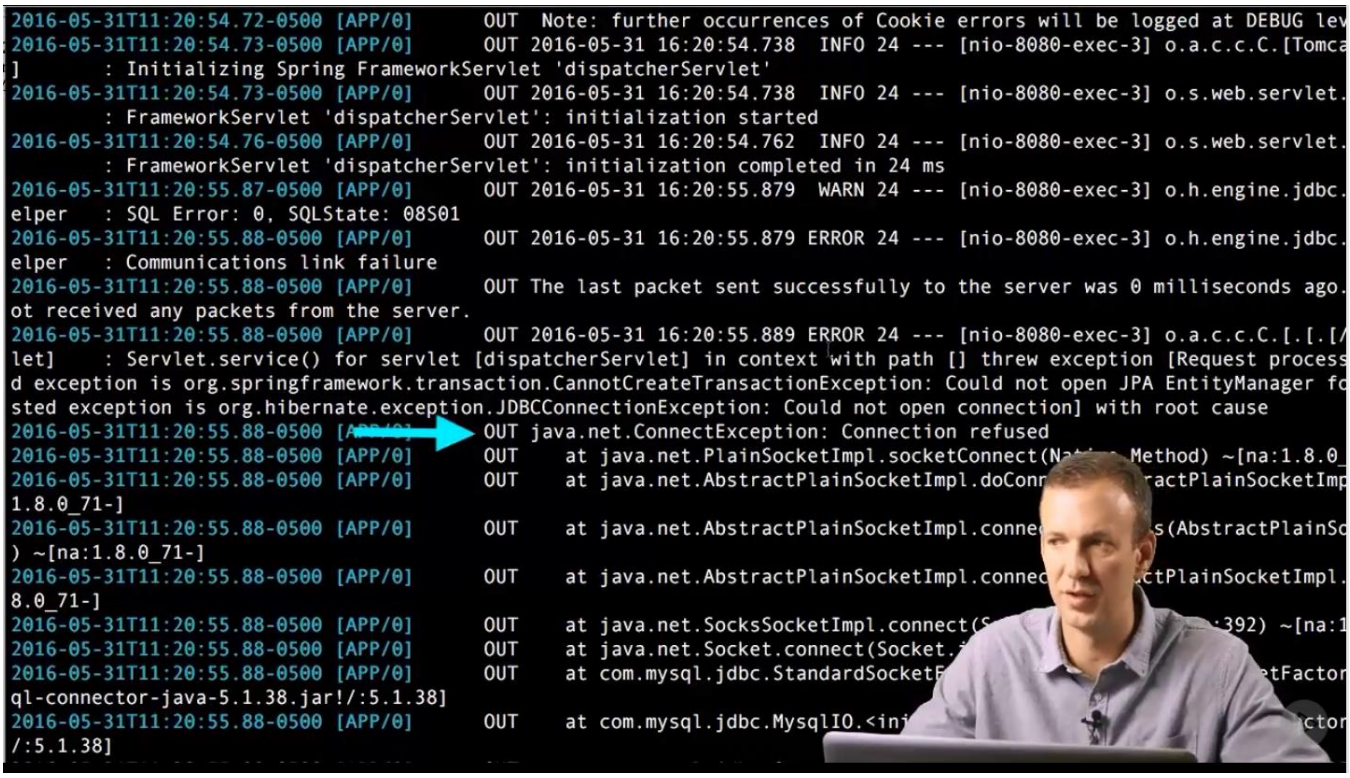
```
DROBERTS-MBPRO:attendee-service droberts$ cf logs attendee-service
Connected, tailing logs for app attendee-service in org dave / space dev as admin...
```

We then start tailing the logs coming out of the attendee-service using the command ***$ cf logs attendee-service***

```
DROBERTS-MBPRO:attendee-service droberts$ cf apsp
'apsp' is not a registered command. See 'cf help'
DROBERTS-MBPRO:attendee-service droberts$ cf apps
Getting apps in org dave / space dev as admin...
OK

name               requested state   instances   memory   disk   urls
articulate         started           1/1         512M     1G     articulate-turbosupercharged-spinneret.cfapps.haas-
attendee-service   started           1/1         512M     1G     attendee-service-monochromatic-guarantee.cfapps.haa
io
DROBERTS-MBPRO:attendee-service droberts$
```



We then call one of the REST APIs on the attendee-service. We can see that we get a JDBC connection error because we cannot open a connection because the SG was unbound and there is no outbound connection for our application.



All the egress traffic into our app has been locked down because we have unbound the **all_open** ASG. We can instead create a space SG that will allow us add some egress traffic to our application for connecting to MySQL.

```
DROBERTS-MBPRO:attendee-service droberts$ cf env attendee-service
Getting env variables for app attendee-service in org dave / space dev as admin...
OK

System-Provided:
{
 "VCAP_SERVICES": {
  "p-mysql": [
   {
    "credentials": {
     "hostname": "10.65.188.80",
     "jdbcUrl": "jdbc:mysql://10.65.188.80:3306/cf_9b4bba91_c2cc_4b82_a9ce_61506b60dce9?user=4otnLyRKEv0bz8r3\u0026p
xqLUjcn",
     "name": "cf_9b4bba91_c2cc_4b82_a9ce_61506b60dce9",
     "password": "lFf1XBQGYxqLUjcn",
     "port": 3306,
     "uri": "mysql://4otnLyRKEv0bz8r3:lFf1XBQGYxqLUjcn@10.65.188.80:3306/cf_9b4bba91_c2cc_4b82_a9ce_61506b60dce9?rec
     "username": "4otnLyRKEv0bz8r3"
    },
    "label": "p-mysql",
    "name": "attendee-mysql",
    "plan": "100mb-dev",
    "provider": null,
    "syslog_drain_url": null,
    "tags": [
     "mysql",
     "relational"
    ]
   }
  ]
 }
```

We can look at the application's environment using the ***$ cf env attendee-service*** command to determine the ports that need to be opened. We can see the application IP address **10.65.188.80** and the MySQL port **3306** that should allow egress traffic, we can then create an ASG file called **asg.json** with this 2 information on our local file system.

```
DROBERTS-MBPRO:attendee-service droberts$ nano asg.json
DROBERTS-MBPRO:attendee-service droberts$
```

```
  GNU nano 2.0.6                          File: asg.json

[
  {
    "destination": "10.65.188.80",
    "ports": "3306",
    "protocol": "tcp"
  }
]
```

```
DROBERTS-MBPRO:attendee-service droberts$ nano asg.json
DROBERTS-MBPRO:attendee-service droberts$ ls
asg.json                              attendee-service-0.0.1-SNAPSHOT.jar
DROBERTS-MBPRO:attendee-service droberts$
```

We can now apply this new ASG within our PCF environment as below

```
DROBERTS-MBPRO:attendee-service droberts$ cf create-security-group mysql-dnr ./asg.json
Creating security group mysql-dnr as admin
OK
DROBERTS-MBPRO:attendee-service droberts$
```

We use the ***$ cf create-security-group mysql-dnr ./asg.json*** command to create a new ASG called **mysql-dnr** from our **asg.json** file in the path specified as above.

```
DROBERTS-MBPRO:attendee-service droberts$ cf create-security-group mysql-dnr ./asg.json
Creating security group mysql-dnr as admin
OK
DROBERTS-MBPRO:attendee-service droberts$ cf bind-security-group mysql-dnr dave dev
Assigning security group mysql-dnr to space dev in org dave as admin...
OK



TIP: Changes will not apply to existing running applications until they are restarted.
DROBERTS-MBPRO:attendee-service droberts$
```

We now use the ***$ cf bind-security-group mysql-dnr dave dev*** command to bind the ASG to the **dave** Org and the **dev** Space.

```
DROBERTS-MBPRO:attendee-service droberts$ cf restart attendee-service
Stopping app attendee-service in org dave / space dev as admin...
OK

Starting app attendee-service in org dave / space dev as admin...

0 of 1 instances running, 1 starting
0 of 1 instances running, 1 starting
1 of 1 instances running

App started


OK

App attendee-service was started using this command `CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_jre/bin/java-
calculator-2.0.1_RELEASE -memorySizes=metaspace:64m.. -memoryWeights=heap:75,metaspace:10,native:10,stack:5 -memoryI
,metaspace:100% -totMemory=$MEMORY_LIMIT) && JAVA_OPTS="-Djava.io.tmpdir=$TMPDIR -XX:OnOutOfMemoryError=$PWD/.java-b
_jre/bin/killjava.sh $CALCULATED_MEMORY" && SERVER_PORT=$PORT eval exec $PWD/.java-buildpack/open_jdk_jre/bin/java $
D/.:$PWD/.java-buildpack/spring_auto_reconfiguration/spring_auto_reconfiguration-1.10.0_RE[illegible] org.springframew
arLauncher`

Showing health and status for app attendee-service in org dave / space dev as admin...
```

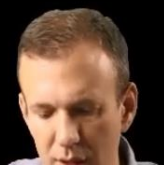
We then restart the application using the ***$ cf restart attendee-service*** command to have all the new security group rules take effect.

attendee-service-monochromatic-guarantee.cfapps.haas-39.pez.pivotal.io/attendees


```
{
  "_embedded": {
    "attendees": [
      {
        "firstName": "David",
        "lastName": "Roberts",
        "address": null,
        "city": null,
        "state": null,
        "zipCode": null,
        "phoneNumber": null,
        "emailAddress": "UBIQUITOUS_ALIAS@YAHOO.COM",
        "_links": {
          "self": {
            "href": "http://attendee-service-monochromatic-guarantee.cfapps.haas-39.pez.pivotal.io/attendees/2"
          },
          "attendee": {
            "href": "http://attendee-service-monochromatic-guarantee.cfapps.haas-39.pez.pivotal.io/attendees/2"
          }
        }
      },
      {
        "firstName": "David",
        "lastName": "Roberts",
        "address": null,
        "city": null,
        "state": null,
        "zipCode": null,
        "phoneNumber": null,
        "emailAddress": "UBIQUITOUS_ALIAS@YAHOO.COM",
        "_links": {
          "self": {
            "href": "http://attendee-service-monochromatic-guarantee.cfapps.haas-39.pez.pivotal.io/atter
```


We can now see that our attendee-service is now allowed to make an outbound connection to MySQL for data over the allowed port and IP.

```
DROBERTS-MBPRO:attendee-service droberts$ cf login
API endpoint: https://api.run.haas-39.pez.pivotal.io

Email> droberts@pivotal.io

Password>
Authenticating...
Credentials were rejected, please try again.

Password>
Authenticating...
OK

Targeted org dave

Targeted space dev



API endpoint:   https://api.run.haas-39.pez.pivotal.io (API version: 2.54.0)
User:           droberts@pivotal.io
Org:            dave
Space:          dev
DROBERTS-MBPRO:attendee-service droberts$
```



We can now login as a developer to see the same SG effect

```
DROBERTS-MBPRO:attendee-service droberts$ cf security-groups
Getting security groups as droberts@pivotal.io
OK

     Name        Organization   Space
#0   mysql-dnr   dave           dev
DROBERTS-MBPRO:attendee-service droberts$
```

We can see the newly applied SG

```
DROBERTS-MBPRO:attendee-service droberts$ cf security-groups
Getting security groups as droberts@pivotal.io
OK

     Name        Organization   Space
#0   mysql-dnr   dave           dev
DROBERTS-MBPRO:attendee-service droberts$ cf security-group mysql-dnr
Getting info for security group mysql-dnr as droberts@pivotal.io
OK

Name    mysql-dnr
Rules
        [
                {
                        "destination": "10.65.188.80",
                        "ports": "3306",
                        "protocol": "tcp"
                }
        ]

     Organization   Space
#0   dave           dev
DROBERTS-MBPRO:attendee-service droberts$
```

The new SG reflects correctly what we defined in our JSON file at creation.

Log Drain

Agenda

1. Cloud Native Apps
2. Loggregator Review

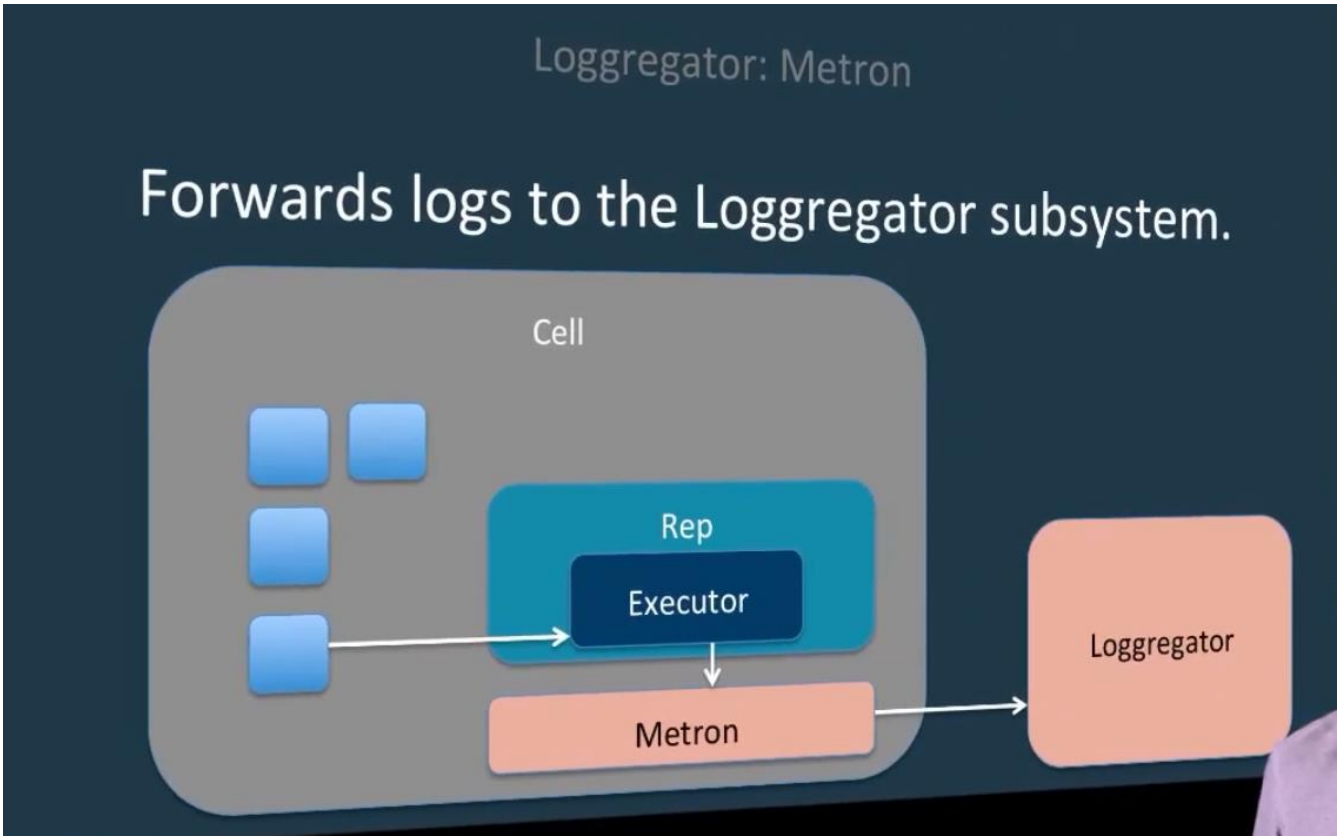
The 12 Factors

Codebase	Dependencies	Config	Backing Services
Build, release, run	Processes	Port binding	Concurrency
Disposability	Dev/prod parity	Logs	Admin Processes

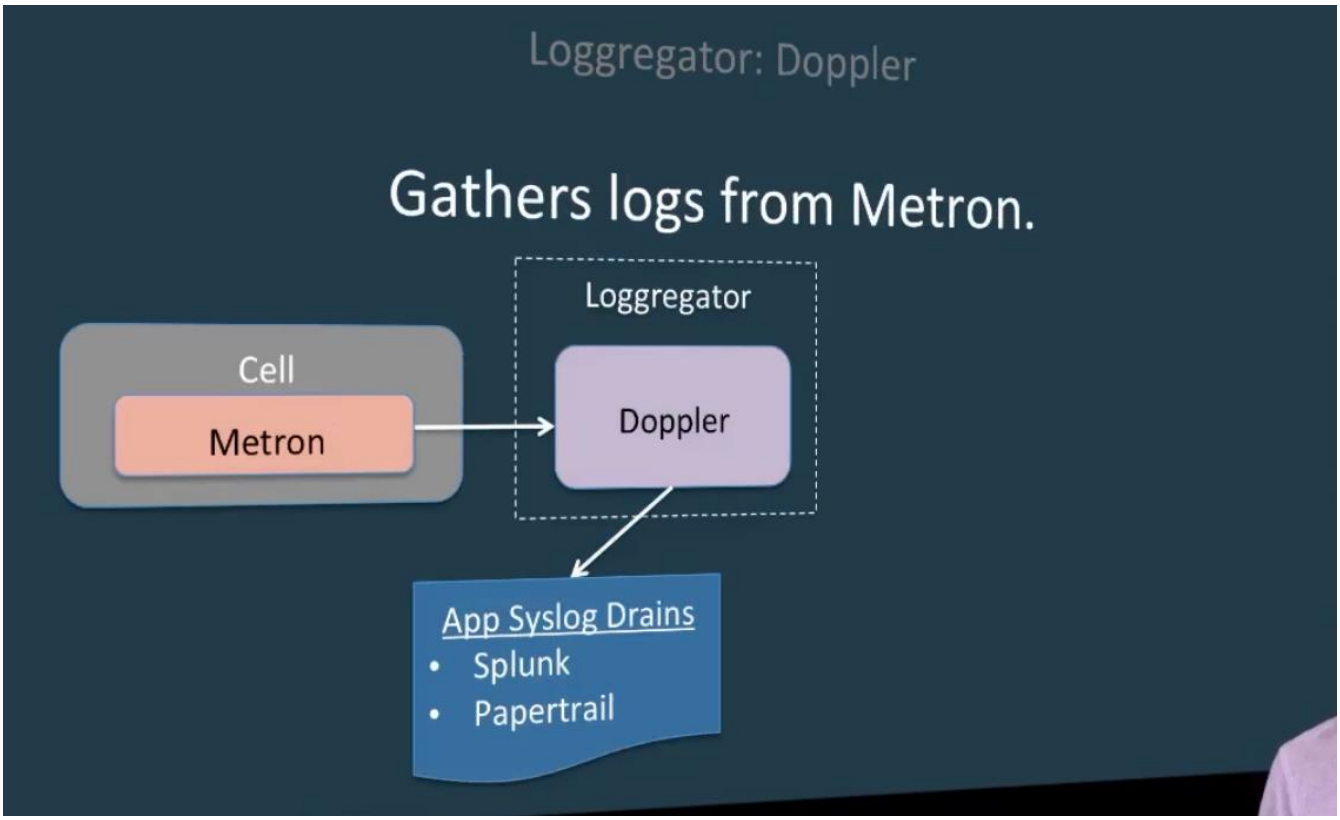
Logs

Treat logs as event streams.

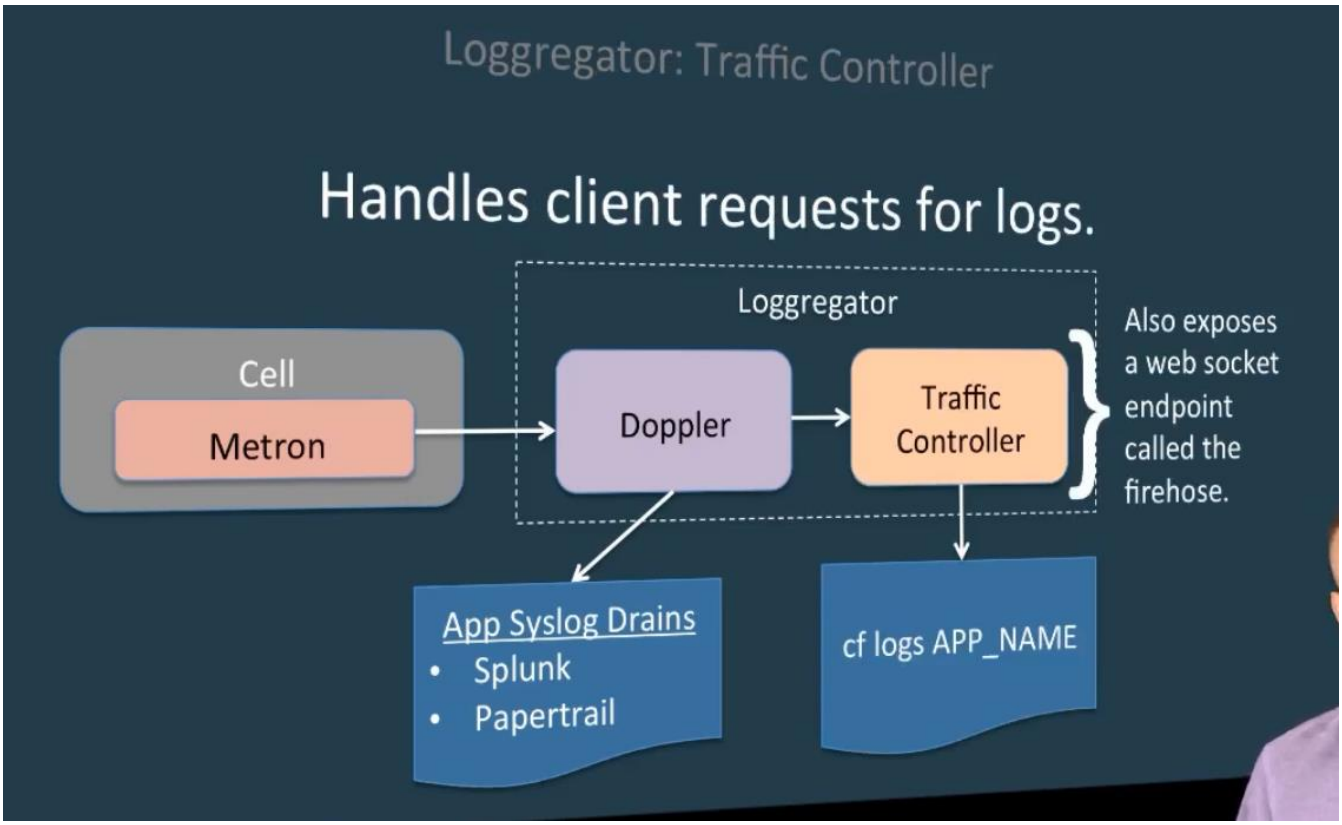
This allows us to easily get our application logs into another system that will allow us investigate the data further, all your app needs to do is to write out the logs to stdout.



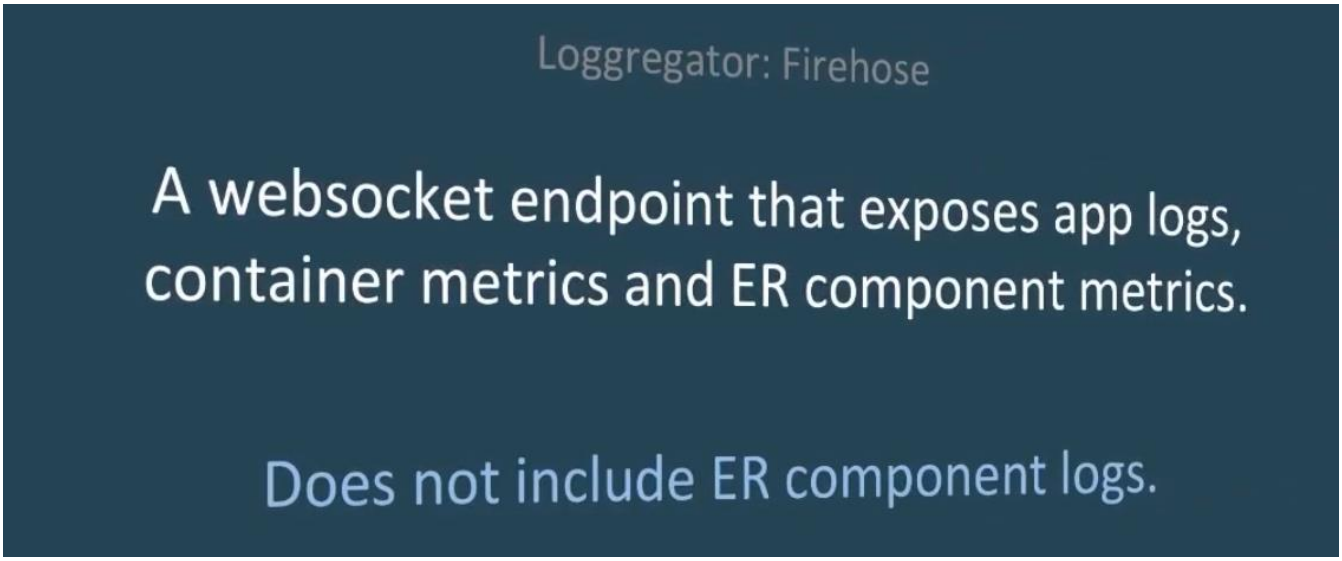
The Executor captures all the stdouts and stderrs from your application streams and sends them to Metorn that forwards them to Loggregator.

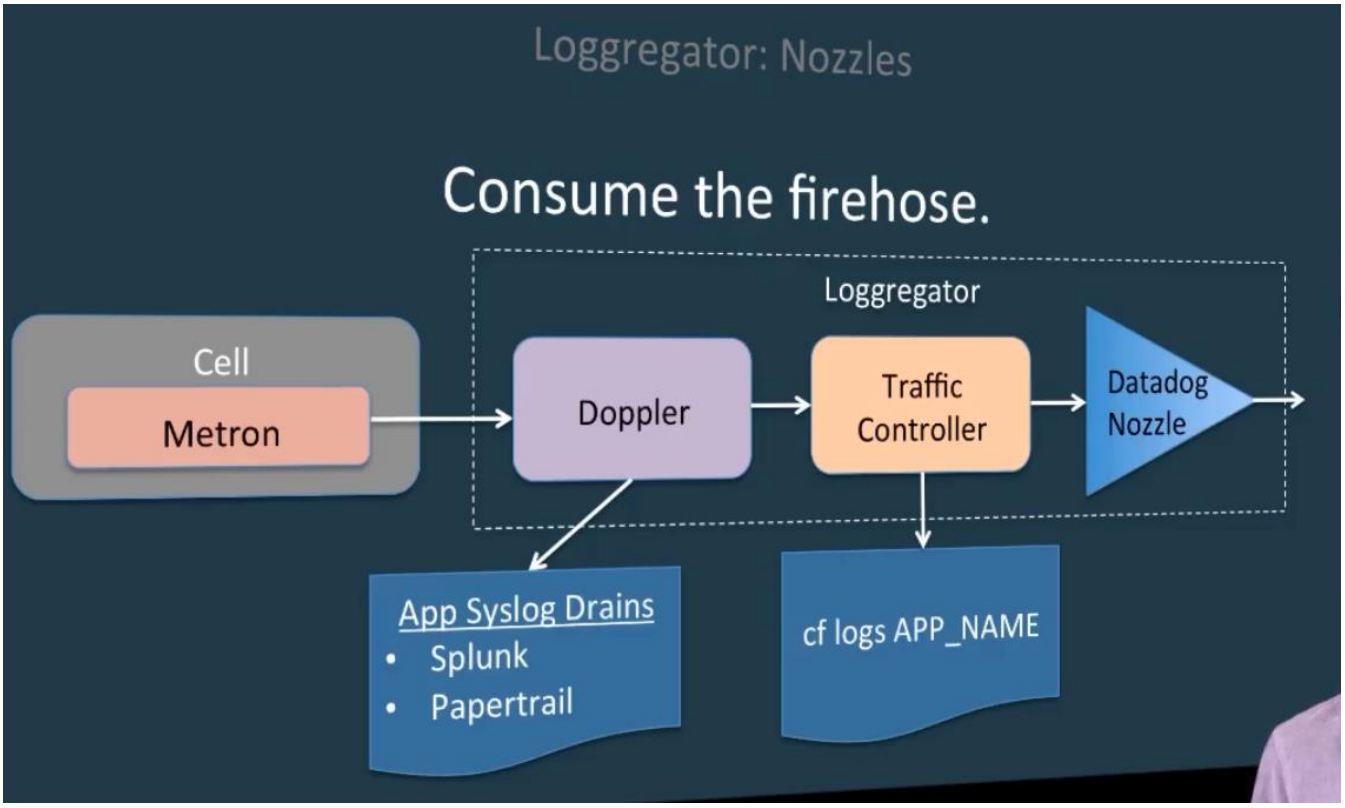


Doppler is responsible for forwarding the logs to the different log drains that you define for your applications.

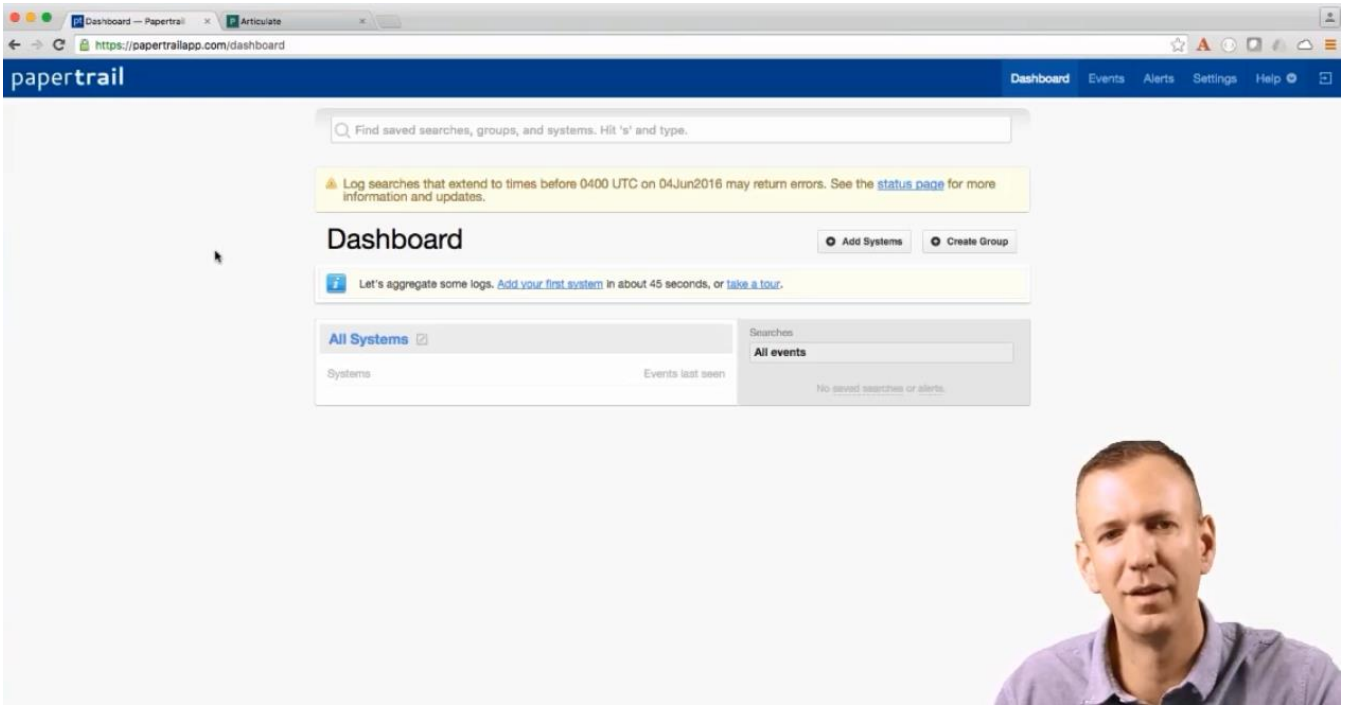


The Traffic Controller is the component that you are interacting with when you run the ***$ cf logs*** command from the CF CLI.

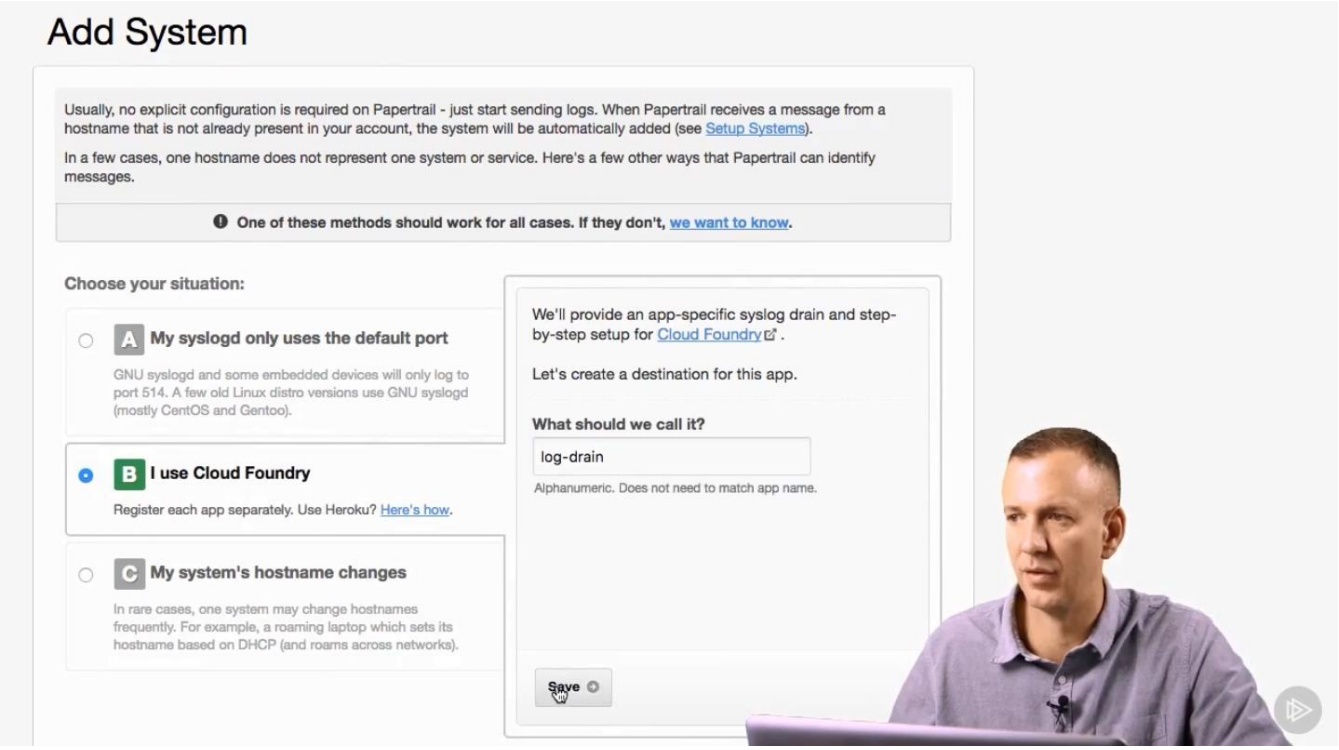


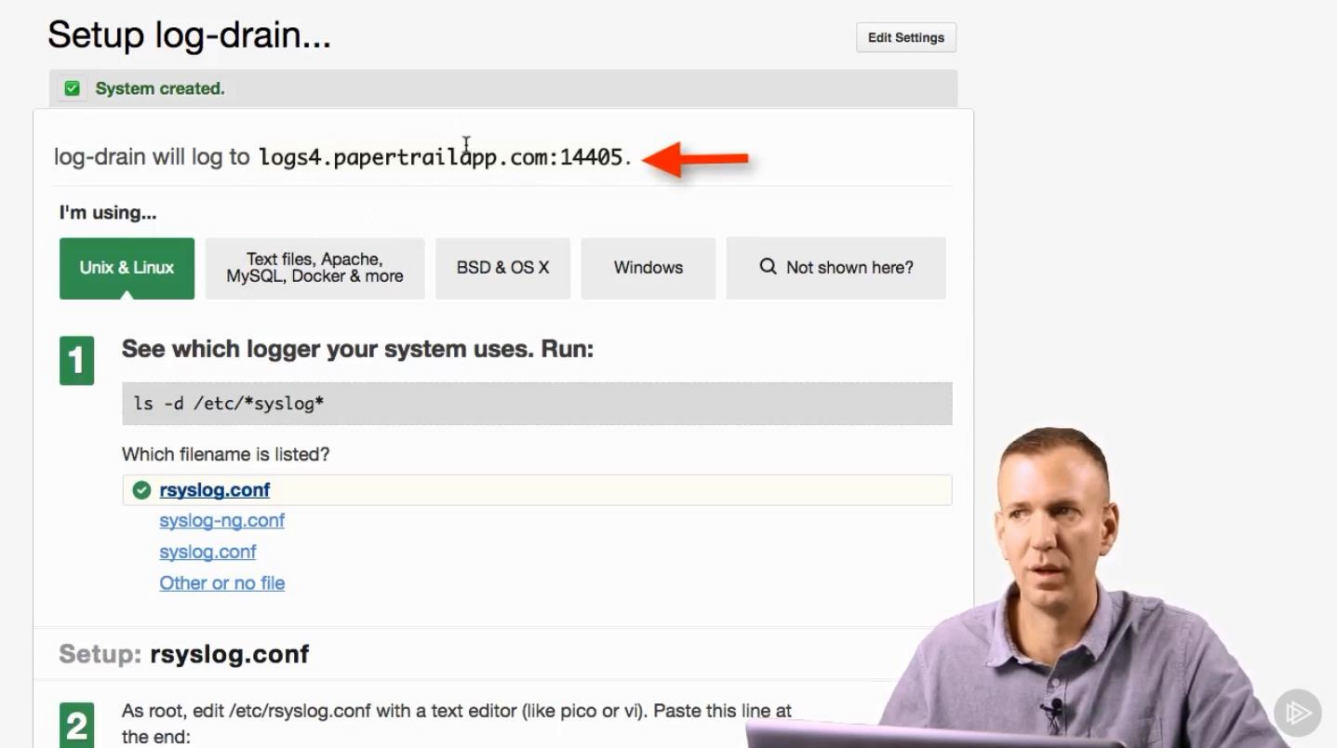


The firehose gets consumed by components called Nozzles. Next, let us see a quick demonstration of a log drain.



We have set up an account with PaperTrail, a log management 3rd party service. We will then set up our app to send logs to PaperTrail.





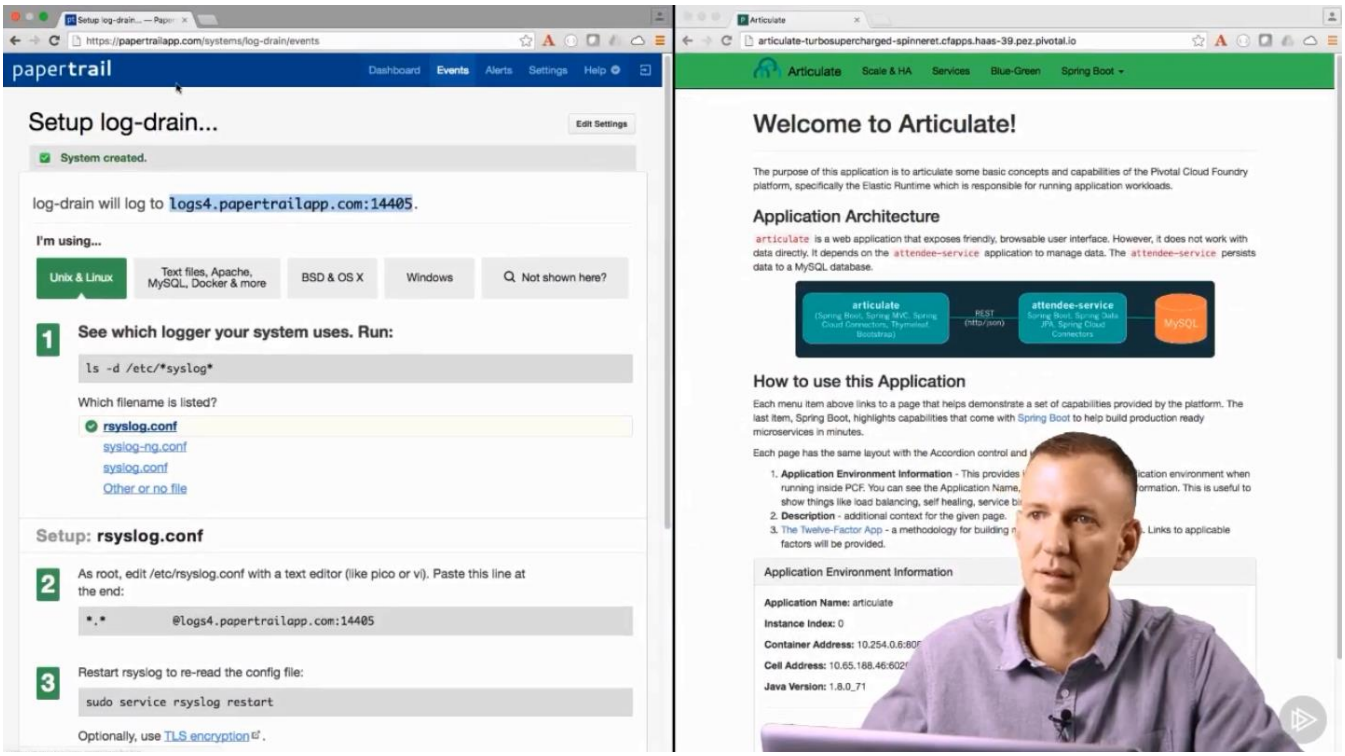
We now have URL to send our logs to from the articulate application.

```
DROBERTS-MBPRO:articulate droberts$ cf create-user-provided-service articulate-log-drain -l syslog://logs4.papertrailapp.
com:14405
Creating user provided service articulate-log-drain in org dave / space dev as droberts@pivotal.io...
OK
DROBERTS-MBPRO:articulate droberts$
```

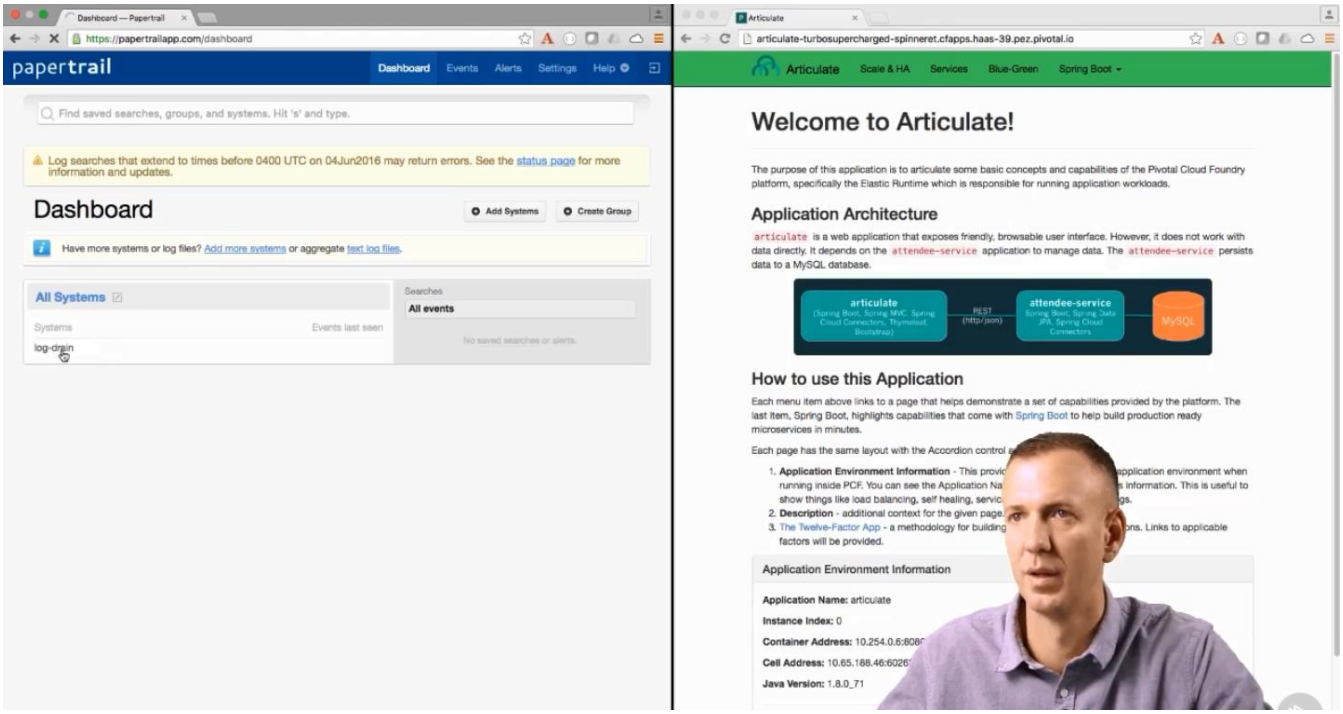
We create our **articulate-log-drain** log-drain service for the articulate application using the command ***$ cf create-user-provided-service articulate-log-drain -l syslog://<our papertrail URL logs endpoint>*** above.

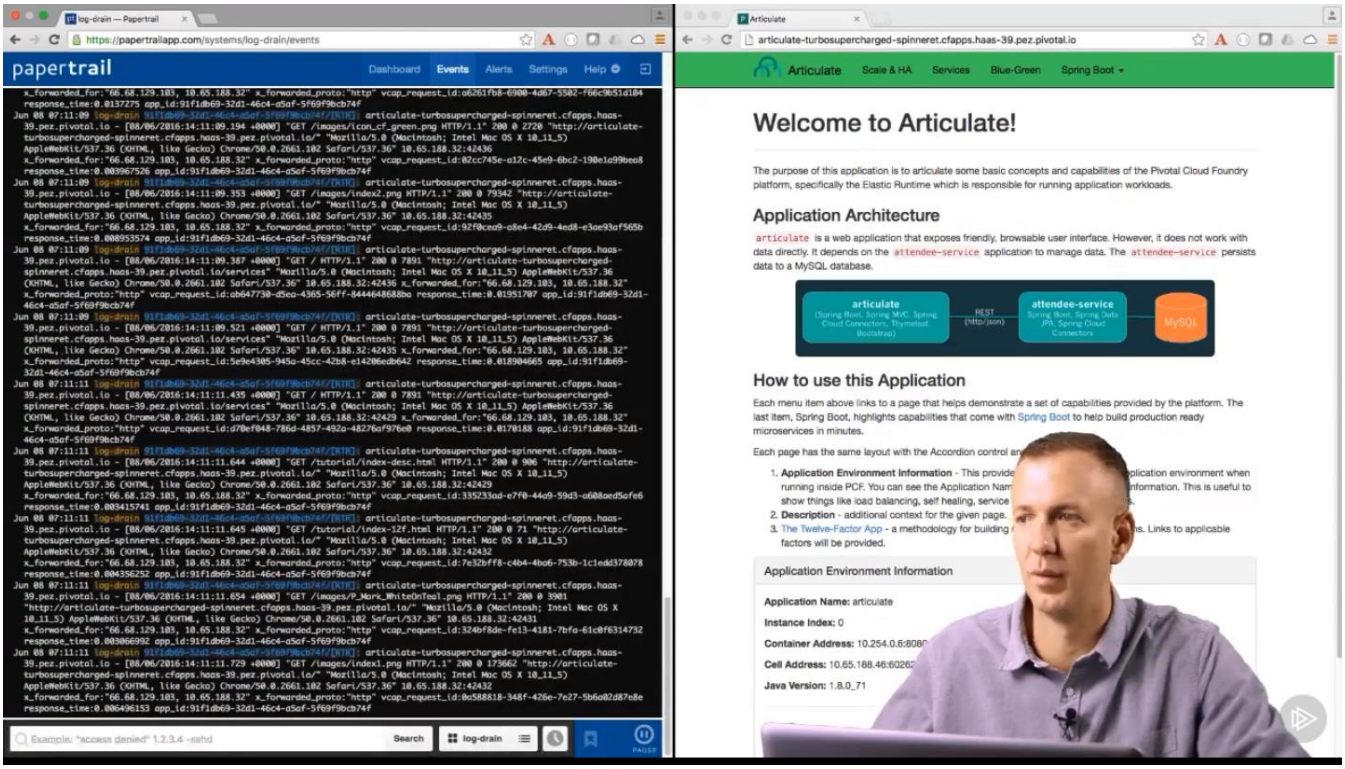
```
DROBERTS-MBPRO:articulate droberts$ cf bind-service articulate articulate-log-drain
Binding service articulate-log-drain to app articulate in org dave / space dev as droberts@pivotal.io...
OK
TIP: Use 'cf restage articulate' to ensure your env variable changes take effect
DROBERTS-MBPRO:articulate droberts$
```

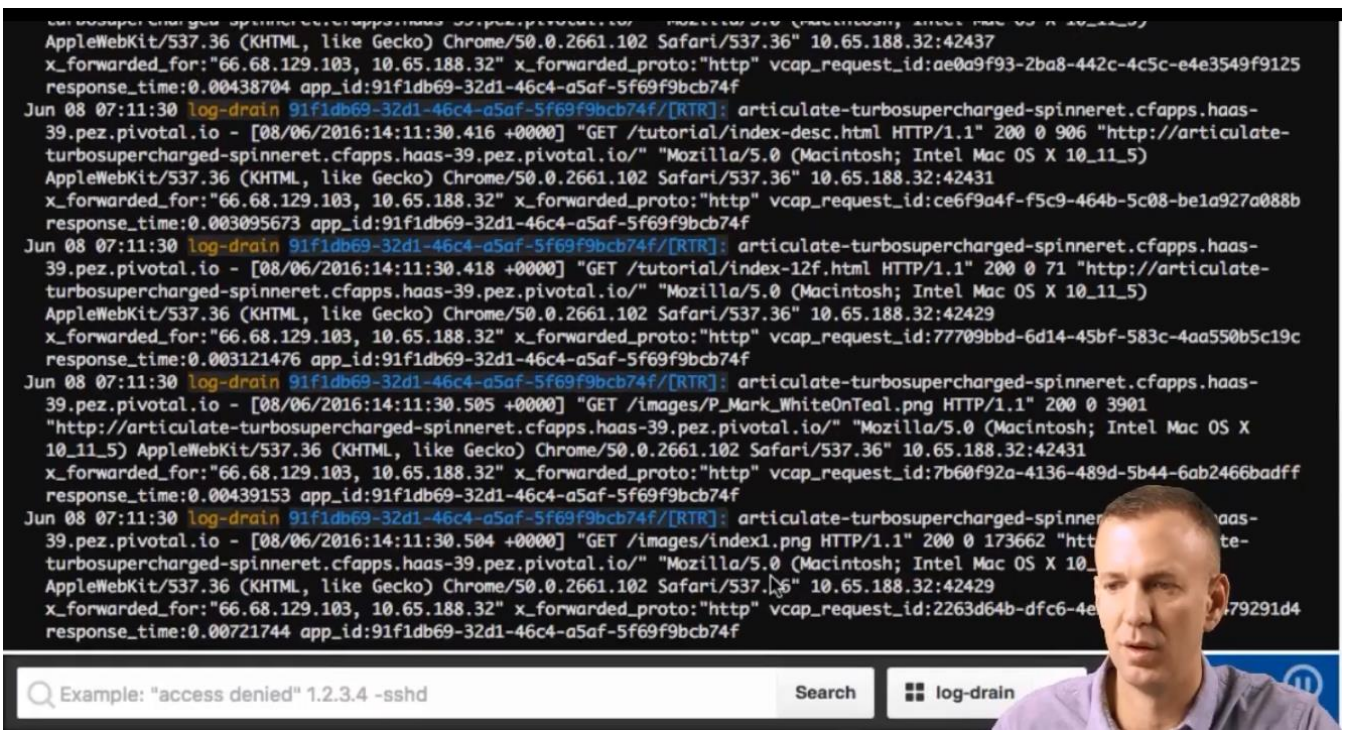
We then bind our new **articulate-log-drain** service to our **articulate** application using the **$ cf bind-service articulate articulate-log-drain** command above. Our points are now going to start flowing into our PaperTrail endpoint for further analysis.



We can then generate some traffic using the articulate application

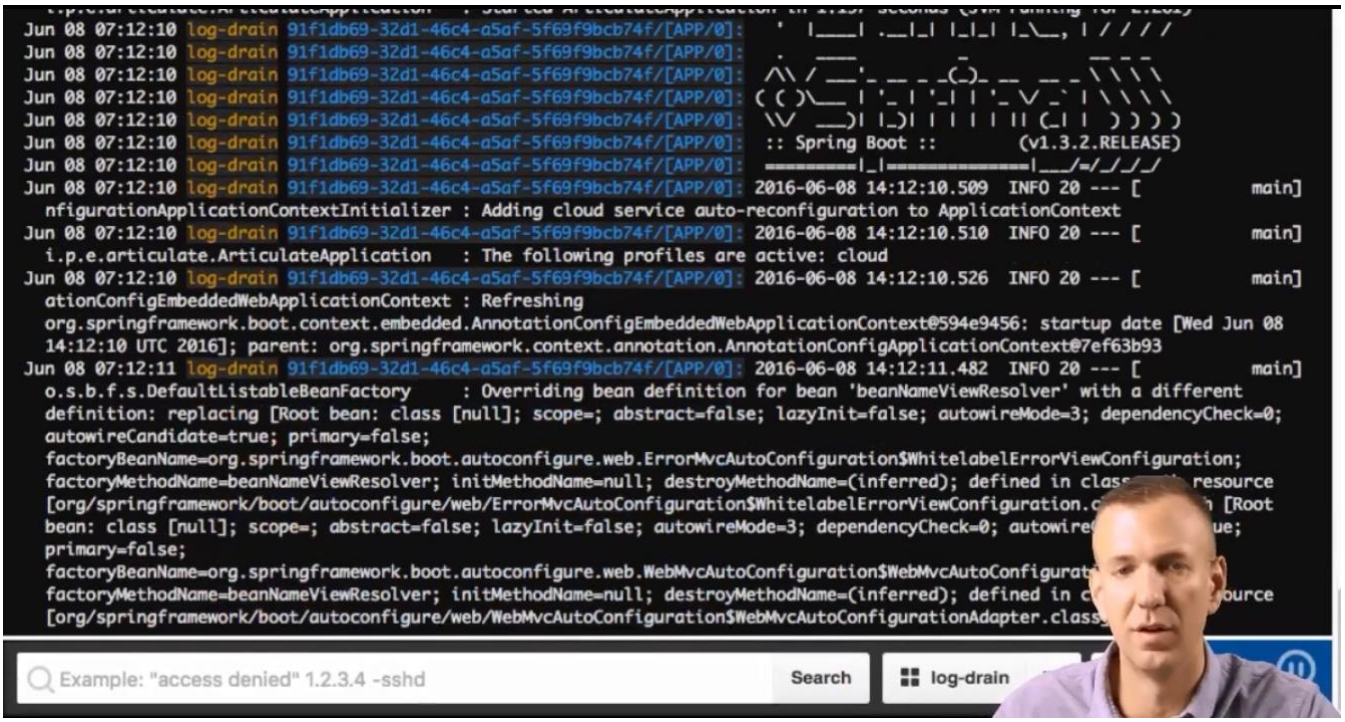




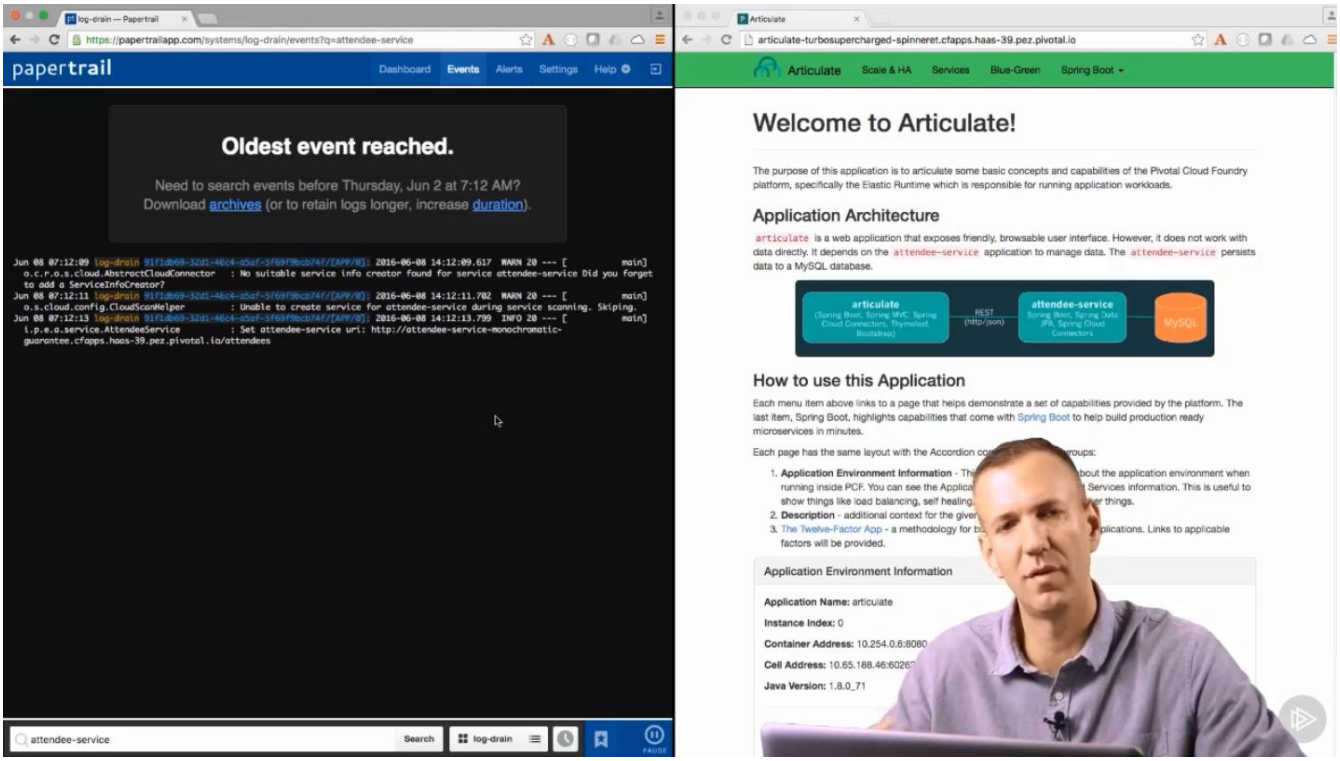


We can now start seeing our logs drained from the articulate application.

```
DROBERTS-MBPRO:articulate droberts$ cf restart articulate
Stopping app articulate in org dave / space dev as droberts@pivotal.io...
OK

Starting app articulate in org dave / space dev as droberts@pivotal.io...

0 of 1 instances running, 1 starting
```



The logs are now flowing into PaperTrail in near real-time.



And we can do specific searches on our logs. We can also use **Splunk** instead of PaperTrail.

Log Drain

Recap