Running PostGIS in Containers At Scale

Steve Pousty

Slides Used in Class

Running autoscaling Postgresql at Scale!

Welcome

Greetings and welcome to the workshop. By the time you leave today you will have gotten your hands nice and dirty playing with OpenShift. My overall goal for today is to:

1. Introduce you to running applications in containers
2. Seeing some cool things you can now have with PostGIS

FYI

- We have 4 hours of class ahead of us
- Use the bathroom when you want
- We are here to help you so please ask questions and interrupt us
- You can keep using this VM when you get home to play with
    1. Docker
    2. Kubernetes
    3. OpenShift

Installing the Vagrant Box

Let's go ahead and install the Vagrant box file and get everything up and running.

Hardware Requirements

1. 10 gigs of disk to start but may need up to 30
2. At least 8 gigs of RAM by default. Our VM will use 4 by default so we want to leave 4 gigs for the OS and others

Software Requirements

WARNING	You must do this **BEFORE** the class - we will not have time to do this in class.

1. Install VirtualBox
2. Install Vagrant

Account requirements

WARNING	You must do this **BEFORE** the class - we will not have time to do this in class.

1. GitHub account, if you don't have one

Github Repos to bring to your machine

These repos may not be finished or even available until right before the class, so we will be doing this in class.

Put these in a directory titled *os_workshop* (not required but then it is on you to translate to your directory name in the rest of the instructions)

FORK The sample code https://github.com/thesteve0/v3simple-spatial and then clone to your machine. If you just clone without the fork you will not be able to do any of the code change exercises.

Clone the Crunchy DB code https://github.com/CrunchyData/crunchy-containers (optionally you can fork and then clone if you think you are going to want to make your own changes).

Getting the Vagrant box on your local machine

In the *os_workshop* directory make a directory titled *vagrant* and change into it.

Using the box on Atlas

If you are doing this at home (which I hope you did before the class) do the following commands

```
C:\_os_workshop\vagrant> vagrant init thesteve0/foss4gna16
C:\_os_workshop\vagrant> vagrant up
```

There is a 3.8 Gig file that needs to be downloaded so it may take a while. After that is download Vagrant will bring everything up.

Manually copying the box off of a USB drive

If you are doing this in the class we are going to manually bring the box to your machine.

WARNING	You need to have access to your USB drive to choose this method to bring up your Vagrant box

Please try to come 15 minutes early to class so we can try to do the copying and installing before the class starts.

Copy the *Vagrantfile* and *workshop.box* file from the USB stick into the *vagrant* directory created above.

Then in that directory do the following commands:

```
C:\_os_workshop\vagrant> vagrant box add --name origin workshop.box
C:\_os_workshop\vagrant> vagrant up
```

Final step

When "vagrant up" is done you should see a message that ends with:

```
==> default: Successfully started and provisioned VM with 2 cores and 4 G of memory.
==> default: To modify the number of cores and/or available memory modify your local
Vagrantfile
==> default:
==> default: You can now access the OpenShift console on:
https://10.2.2.2:8443/console
==> default:
==> default: Configured users are (<username>/<password>):
==> default: admin/admin
==> default: But, you can also use any username and password combination you would
like to create
==> default: a new user.
==> default:
==> default: You can find links to the client libraries here:
https://www.openshift.org/vm
==> default: If you have the oc client library on your host, you can also login from
your host.
==> default:
==> default: To use OpenShift CLI, run:
==> default: $ vagrant ssh
==> default: $ oc login https://10.2.2.2:8443
```

If this doesn't happen raise your hand or, if you are a good student and did this at home, email me. If you see this then you have installed everything properly.

I know some of you are going to be curious and are going to start playing with this VM. That's OK!

Right before class (or if you break it) you just need to do the following steps:

```
C:\_os_workshop\vagrant>vagrant destroy --force
C:\_os_workshop\vagrant>rm -rf .vagrant #basically delete the .vagrant dir - I do it
bc I am superstitious
C:\_os_workshop\vagrant>vagrant up
```

You don't even need to be online for this command to work! Welcome to the future.

Brief Introduction to OpenShift

OpenShift is Red Hat's Container Application Platform. What this means in plain English is OpenShift helps your run Docker containers in an orchestrated and efficient way on a fleet of machines. OpenShift will handle all the networking, scheduling, services, and all the pieces that make a up a modern application.

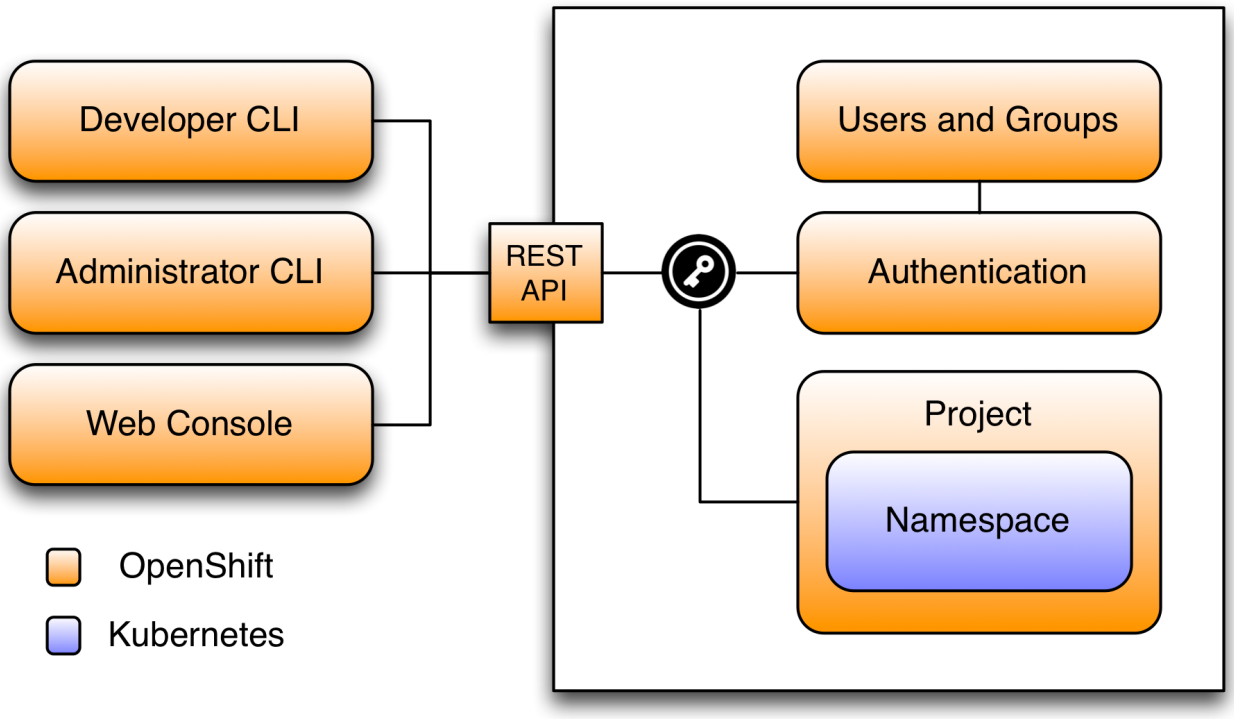
To accomplish this there are required pieces to the platform. We use the Docker container engine to handle just the running of the containers. Above that there is Kubernetes, the Open Source project for orchestrating, scheduling, and other tasks for actually building real applications out of Docker containers. Finally, we layer on top of this the engineering work of OpenShift to build a developer and system adminstrator experience that allows for ease of use when developer and administering containerized applications.

For both Docker and Kubernetes, OpenShift does not do any forking or proprietary extensions. With an OpenShift cluster you can use the OpenShift interfaces OR you can always use the Docker and Kubernetes interfaces. Granted, those interfaces will only support operations possible in that part of the stack.

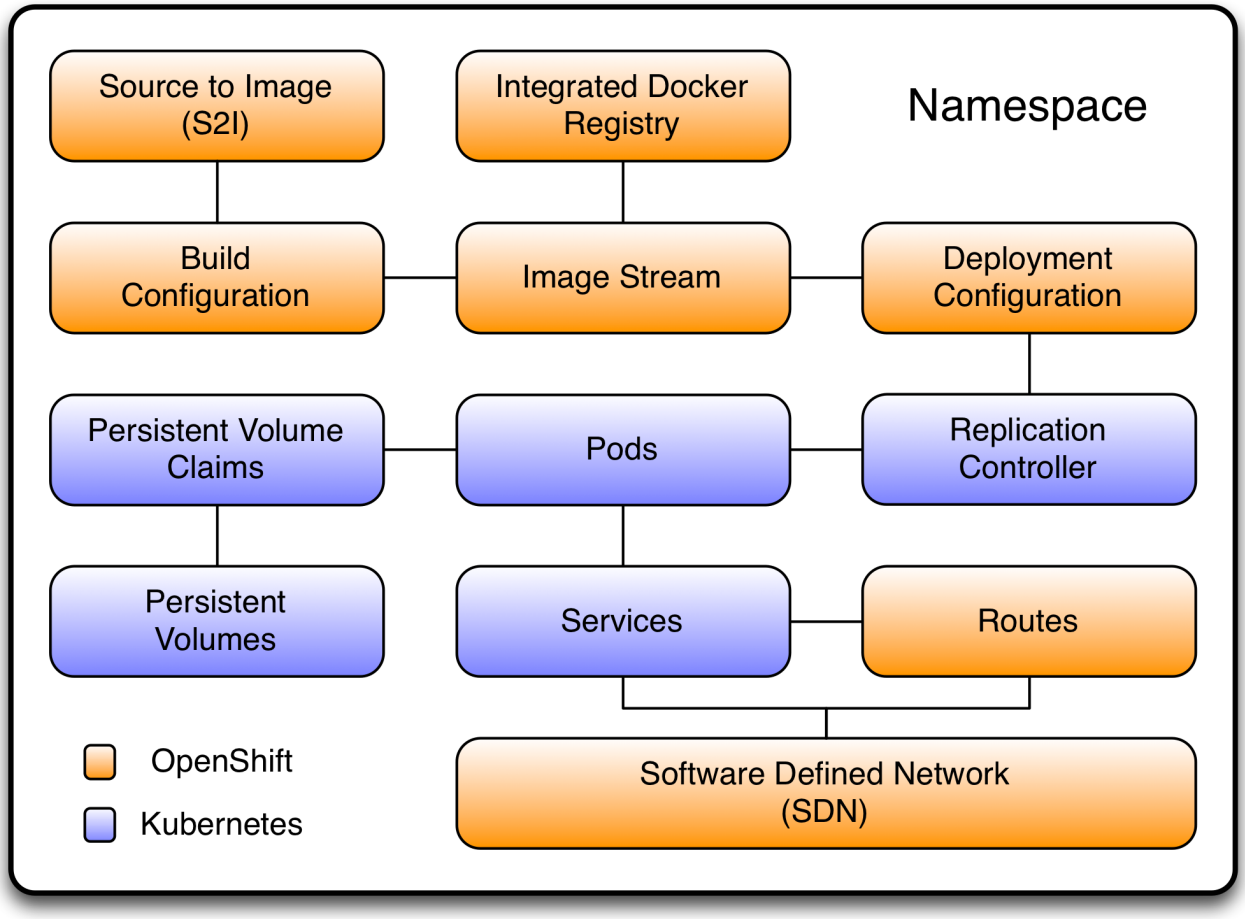
I am going to put two diagrams here that we will be discussing in class. You can refer back to these diagrams while you do the exercices. The key points for these diagrams are:

1. You can look and see how all the pieces fit together
2. You can see which pieces are core Kubernetes (blue) and which pieces are OpenShift (gold).

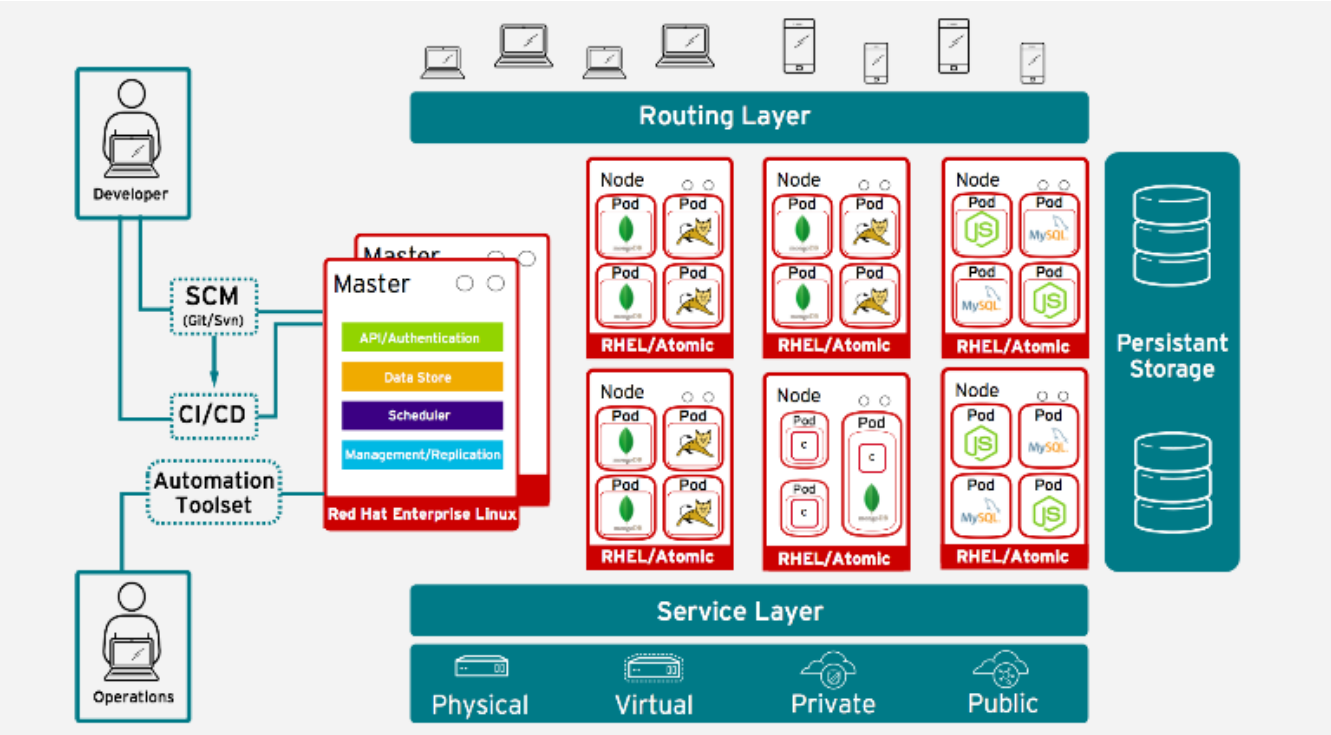
Here is the top level, showing the pieces in the interfaces and authentication:



And here are the lower level pieces that make up the core objects.



Finally, here is a diagram showing how an OpenShift/Kubernetes cluster is put together from different data center "pieces"



Now that we have done enough talking - the rest of the class will be action!

Introduction to the OpenShift All-In-One VM

Before we get started, it is assumed that you have already installed the OpenShift All-In-One VM. If not please refer to [TODO LINK TO PAGE]. It is also assumed you downloaded the OpenShift command line tools and put them in your PATH - if not please refer to [TODO LINK TO PAGE]

The all-in-one VM (A1VM) is intended for a developer or sysadmin to have a quick and easy way to use OpenShift (which also is a great way to use Kubernetes). The focus of the A1VM is:

1. Works right out of the box
2. Relaxes security restrictions so you can run Docker images that run as root
3. Give developers a means to carry out rapid development without needing an external server
4. Give the OpenShift evangelists an easy way to run workshops - like this one

WARNING

We wanted to allow developers to use any Docker image they want, which required us turning off some security in OpenShift. By default, OpenShift will not allow a container to run as root or even a non-random container assigned userid. Most Docker images in Dockerhub do not follow this best practice and instead run as root. Further multiplying this error, a large majority of Dockerhub images are not patched for well known vulnerabilities. Therefore, please use images from Dockerhub with caution. We think some of the risk is mitigated because you are running OpenShift in a VM, but still - be careful which Docker images you run.

The command line

And with that we begin the journey. Our first step will be to login to the VM both from the command line (cli) and the web console (console). Open a terminal and type in:

```
> oc login https://10.2.2.2:844

# on windows or if you get a cert. error do the following

> oc login https://10.2.2.2:844 --insecure-skip-tls-verify=false
```

You should be prompted for a username and passsword. For ease of use, in the workshop, we will use username *admin* password *password*. You can actually use any password you want with the admin account. You can also use any username and password you want. The username will "matter" in that any project created in that username can only be seen by that username - but these are not secure accounts. Remember - the focus of this VM is ease of use.

After logging in you should see the following message

```
Login successful.

You have access to the following projects and can switch between them with 'oc project
<projectname>':

  * default (current)
  * openshift
  * openshift-infra
  * sample

Using project "default".
```

To check our installation let's look at the sample project with the sample application. Let's switch from the *default* project to the *sample* project.

```
> oc project sample
Now using project "sample" on server "https://10.2.2.2:8443".
```

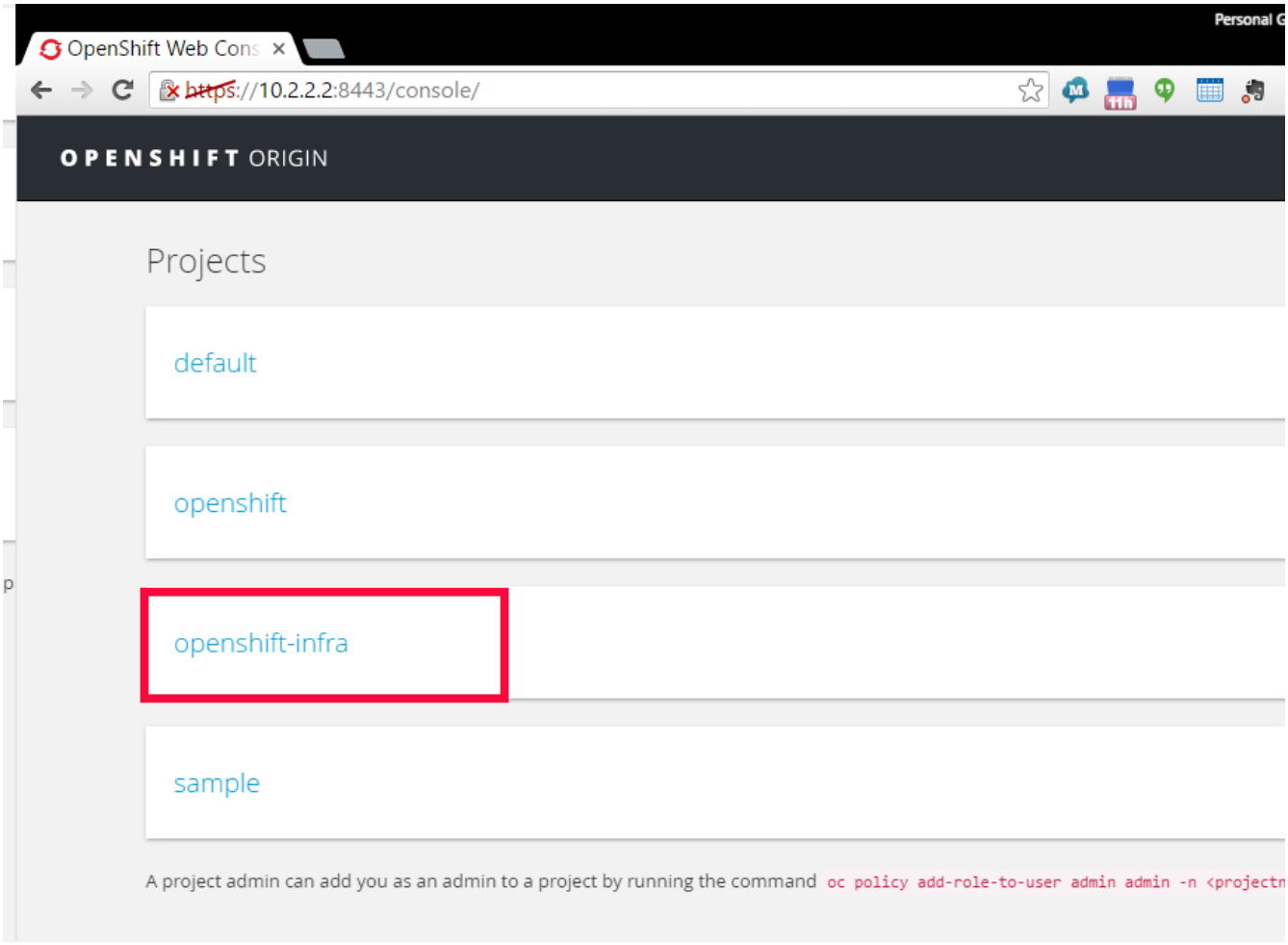
And now let's make sure all our resources are up and running. Run the following command and you should see something like this:

```
> oc status
In project sample on server https://10.2.2.2:8443

svc/database - 172.30.174.52:5434 -> 3306
  dc/database deploys docker.io/openshift/mysql-55-centos7:latest
    deployment #1 deployed 3 days ago - 1 pod

http://frontend-sample.apps.10.2.2.2.xip.io to pod port web (svc/frontend)
  dc/frontend deploys istag/origin-ruby-sample:latest <-
    bc/ruby-sample-build builds https://github.com/openshift/ruby-hello-world.git with
sample/ruby-22-centos7:latest
    deployment #1 deployed 3 days ago - 2 pods

2 warnings identified, use 'oc status -v' to see details.
```

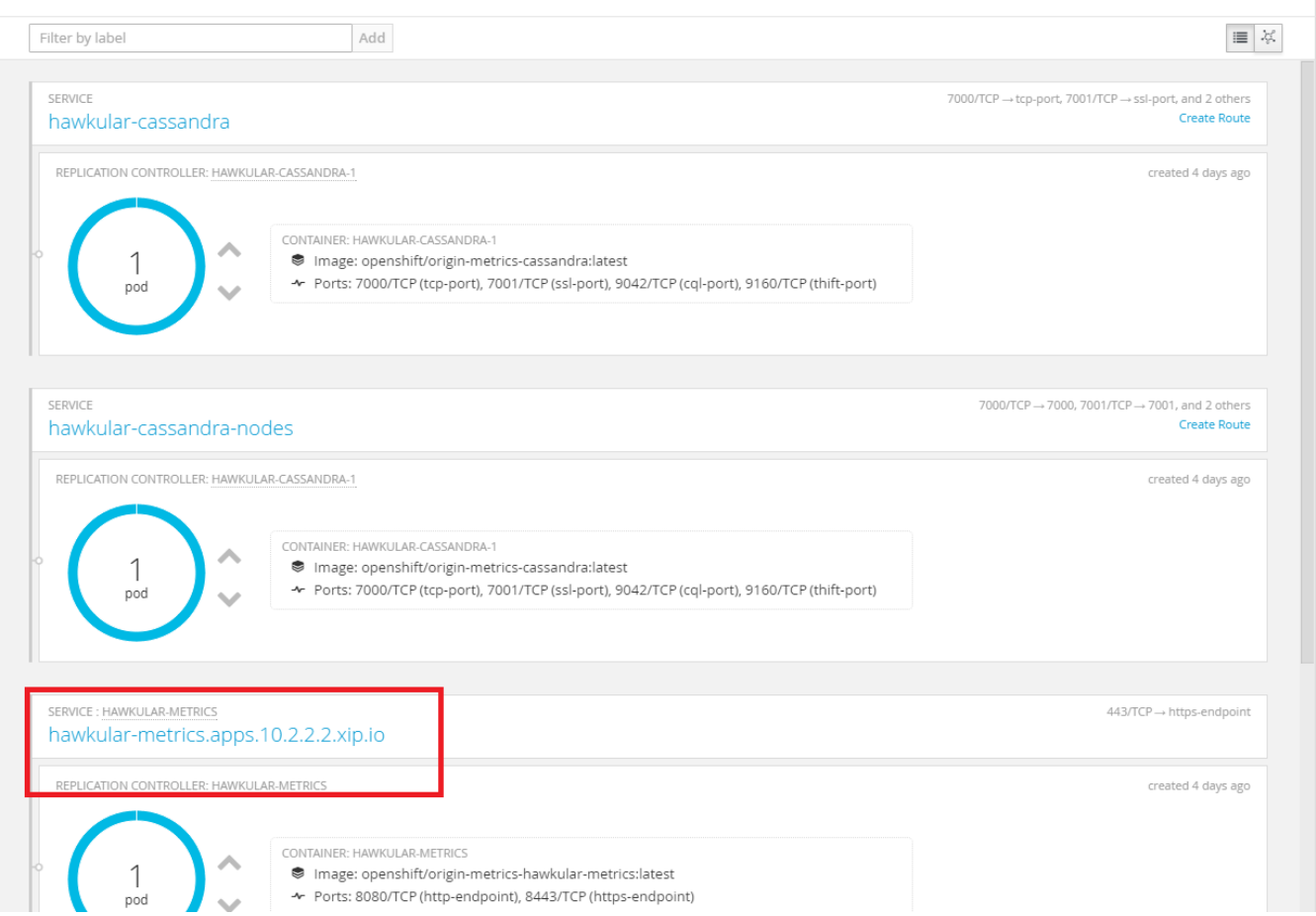
We will revisit explaining all of these terms as we go and build our own application.

Web Console

Now in your browser go to https://10.2.2.2:8443 . You will get a warning about the certificate and this is to be expected since we are using self-signed certificates throughout the installation - so we will need to work around this. You will then get to the login screen. Again use the *admin* and *admin* username password combination and you should see something that looks like this (except for the red box):

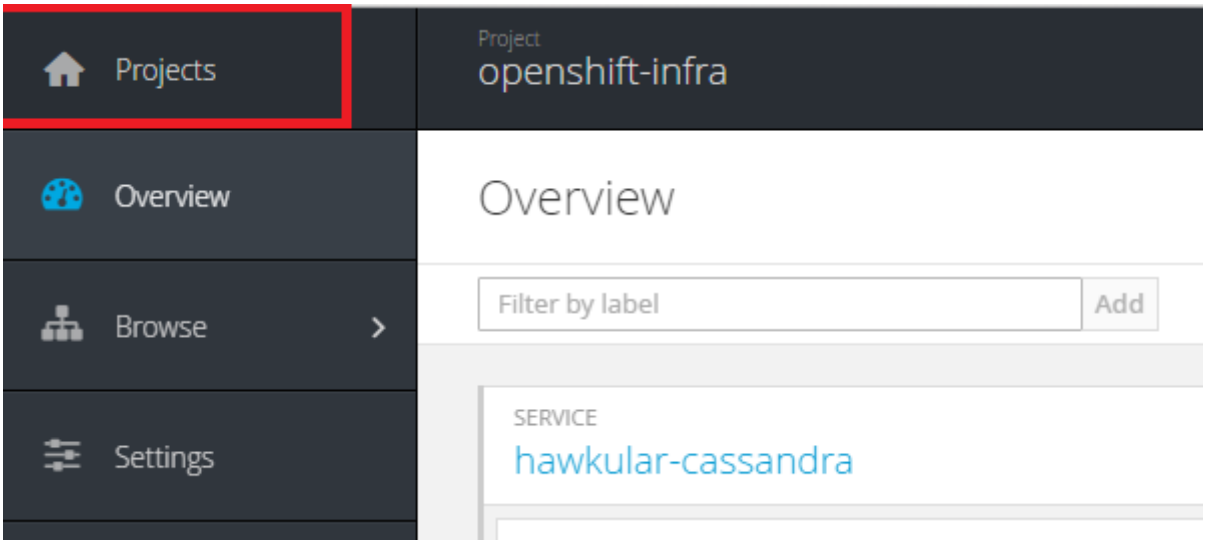


Later in the workshops we will be looking at metrics for our resources and to get that to work we need to accept another self-signed SSL certificate. Please click on the *openshift-infra* project and make sure all the circles are blue. If they are not all blue, they should turn blue in about a minute. If they do not turn blue please raise your hand and call someone over. Now click on the link for https://hawkular-metrics.apps.10.2.2.2.xip.io/ (you can even do it in this document and it should resolve properly)

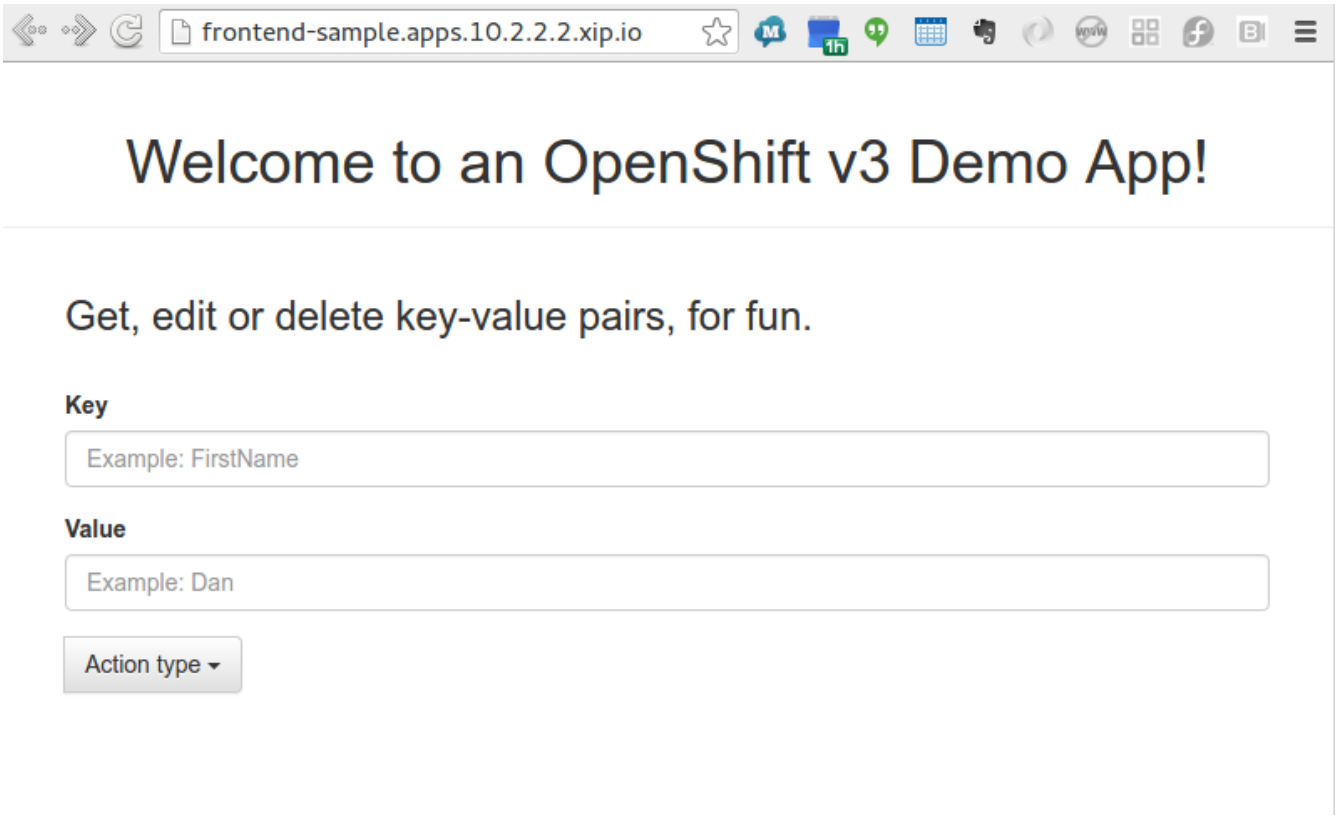


Again, you will be confronted with a invalide cert warning and go ahead go through to the Hawkular page. Now you have the cert stored for that page so the metrics will be viewable.

Now go back to the Projects page by clicking on the house with the word *Projects* next to it in the top right corner of the page.



Now click on the *sample* project and then click on the URL on the page that goes to http://frontend-sample.apps.10.2.2.2.xip.io/ You should see a screen that looks like this:



TIP

There could be several reasons why the URL doesn't resolve but the most common is that something is blocking the xip.io DNS resolution. I have seen several reasons for this: 1. The network is blocking alternate DNS resolution (very common on corporate networks) 2. You have a DNS set as your DNS provider on your machine or your router that blocks xip.io. Some people have had luck switching their DNS provider to Google - 8.8.8.8 & 8.8.8.4 DNS servers

If both of theose steps then we are all set and we can actually move on to you getting some development done.

Beginning Your Application

Let's go ahead and start our own project using Python. I chose Python for this workshop because I thought it would be easiest for you folks to understand regardless of your coding background. The development pattern you are going to use today will apply to any language you use for development in OpenShift.

Bringing in the Code

I am going to start with the command line but we will be moving back and forth between the command line and web UI today.

Making a project

First we are going to make a new project for today's workshop. Projects are a "place" to put together all the pieces you want to logically group. This is usually a single application but it actually could be mutliple applications with multiple URLs.

At your command prompt go ahead and:

```
> oc new-project spatialapp
Now using project "spatialapp" on server "https://10.2.2.2:8443"...
```

That's it - you now a project "spatialapp" and all the subsequent commands will affect this project until you swith to another project. To see all the projects that you have permissions to see go ahead and:

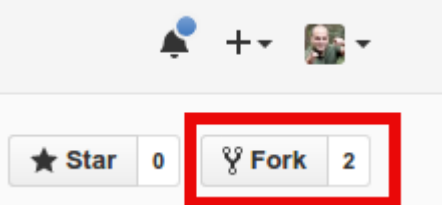
```
> oc get projects
```

Adding Code and Doing a Build

The source code for this project can be found here:

https://github.com/thesteve0/v3simple-spatial

Now **BEFORE** you go and *git clone* it down to your machine, you need to fork it. See that button up towards the top right that says fork on it - click it (you did login with your github account right?)



Now you should be looking at URL that replaces *thesteve0* with your github userid:

https://github.com/<your_userid>/v3simple-spatial

In the middle of the page you will see a button that may say **SSH** on it. If it does, please change that to HTTPS and then click the clipboard to copy the URL in the box.

Our code is ready. Let's fire off the build and deploy. In your terminal enter the following command:

```
> oc new-app python:3.3~https://github.com/thesteve0/v3simple-spatial.git
--> Found image 596ac47 (11 days old) in image stream "python" in project "openshift"
under tag "2.7" for "python:2.7"

    Python 2.7
    ----------
    Platform for building and running Python 2.7 applications

    Tags: builder, python, python27, rh-python27

    * A source build using source code from https://github.com/thesteve0/v3simple-
spatial.git will be created
      * The resulting image will be pushed to image stream "v3simple-spatial:latest"
    * This image will be deployed in deployment config "v3simple-spatial"
    * Port 8080/tcp will be load balanced by service "v3simple-spatial"
      * Other containers can access this service through the hostname "v3simple-
spatial"

--> Creating resources with label app=v3simple-spatial ...
    imagestream "v3simple-spatial" created
    buildconfig "v3simple-spatial" created
    deploymentconfig "v3simple-spatial" created
    service "v3simple-spatial" created
--> Success
    Build scheduled, use 'oc logs -f bc/v3simple-spatial' to track its progress.
    Run 'oc status' to view your app.
```

What we just did is told OpenShift - take this Docker Image that knows how to build a standard Python application layout and combine it with this source code and produce a new Docker Image. As part of this process the *new-app* command knows we will need some other OpenShift objects to

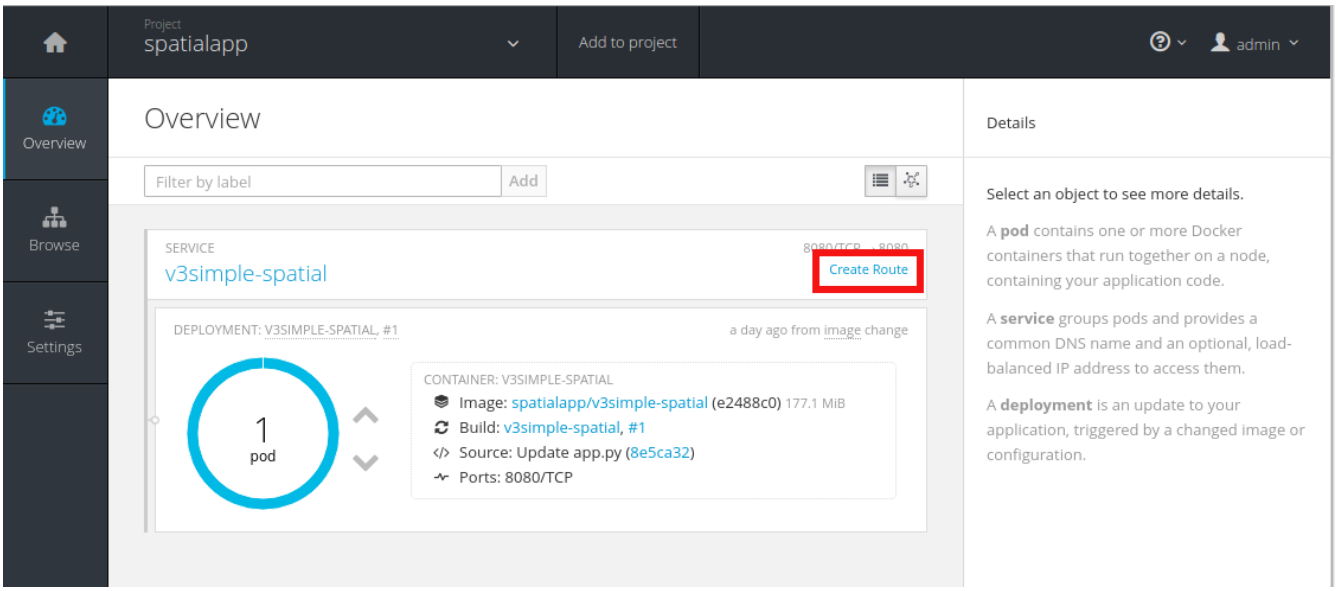
actually run that resulting image. The command goes ahead and makes those objects as well.

TIP	TAKE A BREAK RIGHT HERE BECAUSE I AM GOING TALK YOU THROUGH ALL THOSE OBJECTS IT JUST BUILT. YOU WILL NEED THE DIAGRAMS FROM THE "INTRODUCTION TO OPENSHIFT" TO REFER TO.

Looking at what we built.

At this point we are going to switch back to the web terminal since it is easier to look at web sites in a browser. Go ahead and go into your browser and go to the projects page. Once you are looking at all your projects go ahead and click on *spatialapp*.

Your screen should look something like this:



You can see that we have the 1 pod running our image that was derived from our source code and our builder image. There is also other metadata in the box for the pod that we can return to later. For now I want you to click on the *Create Route* button that is highlighted in red. This will create a URL where we can see the web page for our pod. By default, nothing is exposed to the outside world and you have to choose to expose it.

Just click *Create* on the next page that comes up - all the defaults are fine for our us. You could have also done this at the command line with

```
> oc expose service v3simple-spatial
```

Now when you come back to the Overview page there is a URL for the service ending in .xip.io, go ahead and click it! You will be greated by the following amazing web content:

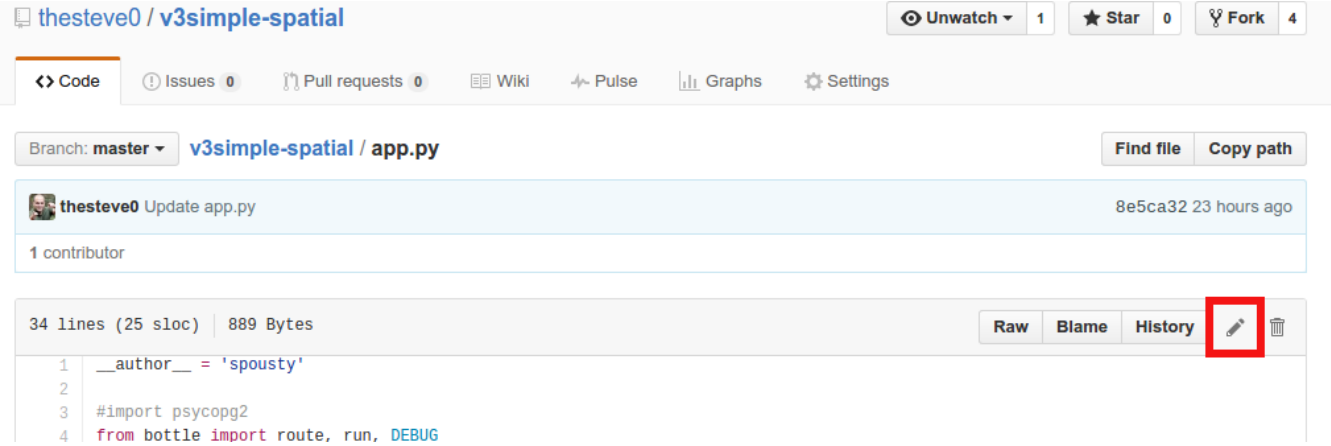
hello OpenShift Ninja without DB

You have now all built and deployed containers with a working URL - give yourselves a pat on the back.

Doing Code Changes With a Build

While that web page is AWESOME, you might actually want to change it. OpenShift gives you the ability to change the code in a "hot swap" fasion for rapid iteration development or in a more complete build cycle which creates a new image. For now we will do the more complete build cycle and the "hot swap" later.

Go ahead to your GitHub fork and edit the app.py file right there in GitHub. When looking at the GitHub repo., click on app.py and then on the next page click on the pencil icon in the top right.



Then on line 11 change the text to whatever you would like. I recommend something like "Steve is the best instructor EVER!" but your tastes may vary (be wrong).

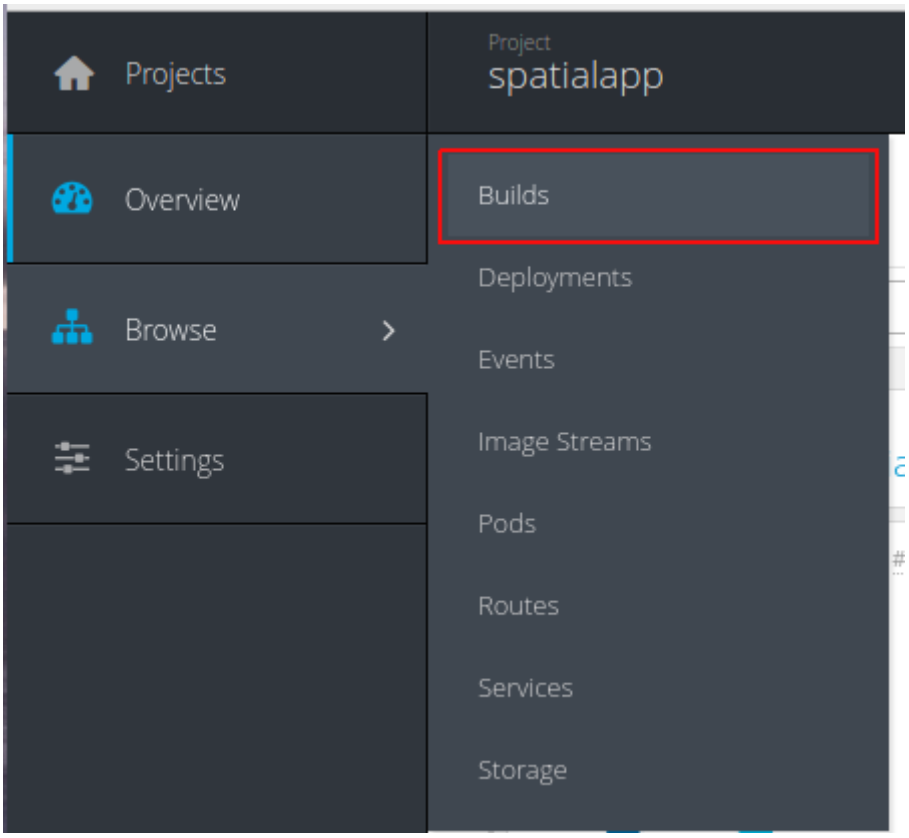
After you are done with your change go ahead and scroll down the page. If you want to add comments or something to your commit go ahead, but everyone needs to press the big GREEN button. You have now committed your change to your GitHub repository.

If our Vagrant image had an IP that was publically accessible, we could actually set up a web hook trigger in GitHub to tell OpenShift to build on commit to master. Instead we have to fire off the build manually. There are two ways to do this:

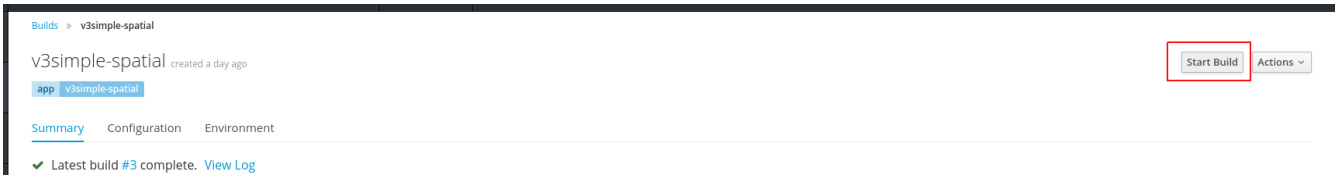
1) At the command line you can do:

```
#get a listing of the build configs (bc)
> oc get bc
NAME               TYPE      FROM      LATEST
v3simple-spatial   Source    Git       1

> oc start-build v3simple-spatial --follow --wait
```

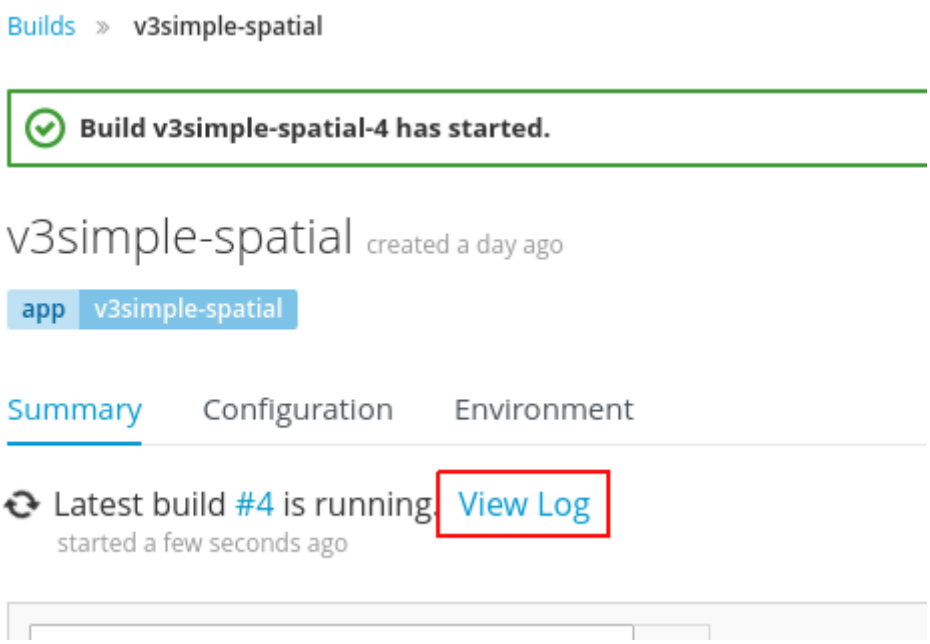
2) You can also use the web console. Click on Browse in the left menu and then choose the builds.



There will only be one build listed on the page, go ahead and click it. Then on the next page in the upper right you will see a button "Start Build", go ahead and click it.



On the next page you can look at the what is happening by clicking on the "View Log" link.



In the terminal on the resulting page you can watch the entire build and push progress: just like you could in the command line with the --follow flag.

With either the command line or the web console, when you go back to the Overview for your project you will notice that the number next to the deployments has incremented because we have actually done a build AND deployment as part of this process.

You are now certified in the Pousty School of Docker and Cloud management as having completed build and deploy MASTERS!

In the next section we will now move on to add PostGIS to your application.

Adding a Database

Now it's time to get to the part of the workshop you have all been waiting for - getting PostGIS up and running! Before we do this though I need to explain just a bit more about how OpenShift works.

All of this work we have done so far has created entries in a database internal to OpenShift (stored in etcd to be exact). You can actually get OpenShift to give you back this information in YAML or JSON format. You can then take that information and give it to another OpenShift instance and get the exact same infrastructure. You can also edit or distribute that JSON or YAML to other people as configuration for their applications. We call these JSON or YAML files templates.

There is also no need to give all the JSON or YAML, you can just distribute pieces of the "infrastructure". The other cool thing you can do is add parameters to the file that can be autogenerated at time of ingestion. These paramters can also be used throughout the same file, insuring that all pieces - like a DB and an app server - get the same values for a setting.

Let's go ahead and look at the JSON for the stuff we have created so far. In the terminal go ahead and do:

```
> oc get svc,dc,bc -o json
```

This will give you the JSON for the services, deploymentConfigs, and buildConfigs you have already created. I will quickly talk through some of the what we are seeing but I, in no way, intend to do a deep dive in this class.

Generating the Database Pieces

In the terminal, please go to the location on your machine where you cloned the Crunchy Solutions repository. Go to the *examples/openshift* directory. We are going to use the file *master-slave-rc-pg-env-vars.json* and you can open it in your favorite text editor now. I am not expecting you to understand it but I want to show you the parameters. Now that we are done talking about it let's do it:

```
> oc new-app .\master-slave-rc-pg-env-vars-with-collect.json
```

That's it - thanks to the work by Jeff you now have a master-replica PostGIS database setup. You can see it in the web overview for the project now. I will talk through the pieces in class.

Loadind the Database

Let's go ahead and load up the database in the master with some DDL. In the v3simple-spatial repository https://github.com/thesteve0/v3simple-spatial.git there is already a SQL file with all our DDL statements. Please clone the repository and then change into the root of the repository.

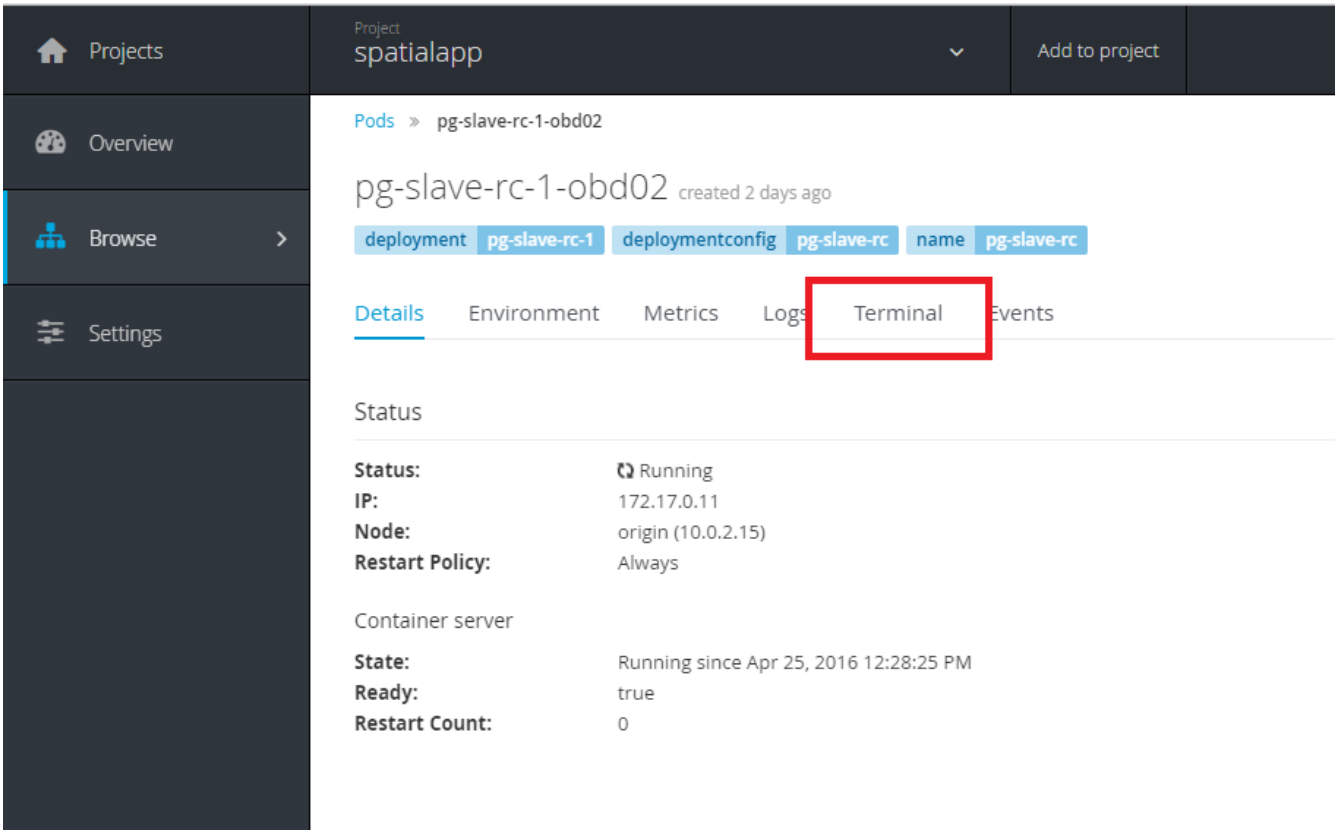
```
# we need to get the master pod - it will have a different name on your machine
> oc get pods
NAME                    READY     STATUS    RESTARTS   AGE
pg-master-rc-1-abvxb    1/1       Running   0          2d
pg-slave-rc-1-obd02     1/1       Running   0          2d

#Look for the conainer named server - we will need this below
> oc describe pods pg-master-rc-1-abvxb

 oc rsync ./ddl pg-master-rc-1-abvxb:/tmp/. -c server
 ddl/parkcoord.sql
```

You may receive a warning that rsync is not found on your machine but the command line tool will fall back to other methods to try and copy the files over. We have now put the DDL file in */tmp/ddl* in the Master postgis pod.

Let's shell into that same pod and load the data

```
> oc rsh -c server pg-master-rc-1-abvxb
sh-4.2$ psql -l
                              List of databases
   Name    |  Owner   | Encoding  | Collate | Ctype |   Access privileges
-----------+----------+-----------+---------+-------+-----------------------
 postgres  | postgres | SQL_ASCII | C       | C     |
 template0 | postgres | SQL_ASCII | C       | C     | =c/postgres          +
           |          |           |         |       | postgres=CTc/postgres
 template1 | postgres | SQL_ASCII | C       | C     | =c/postgres          +
           |          |           |         |       | postgres=CTc/postgres
 userdb    | postgres | SQL_ASCII | C       | C     | =Tc/postgres         +
           |          |           |         |       | postgres=CTc/postgres+
           |          |           |         |       | testuser=CTc/postgres
(4 rows)

sh-4.2$ psql -f /tmp/ddl/parkcoord.sql  userdb
CREATE TABLE
CREATE INDEX
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1...

sh-4.2$ psql userdb
userdb=# select count(*) from parkpoints;
 count
--
   547
(1 row)

userdb=# \q
```

You have now loaded your database with a bunch of points for national parks in the US and Canada. The really amazing part comes next. Go ahead and go to the overview for the project. Go ahead and click on the circle for the *pg-slave-rc* which will bring you to a listing with a single pod. Go ahead and click on that link. On the page for the pod, click on the terminal tab:



On the resulting page you need to click twice on the terminal area to give it focus BUT you are now in terminal in the running pod - slick.

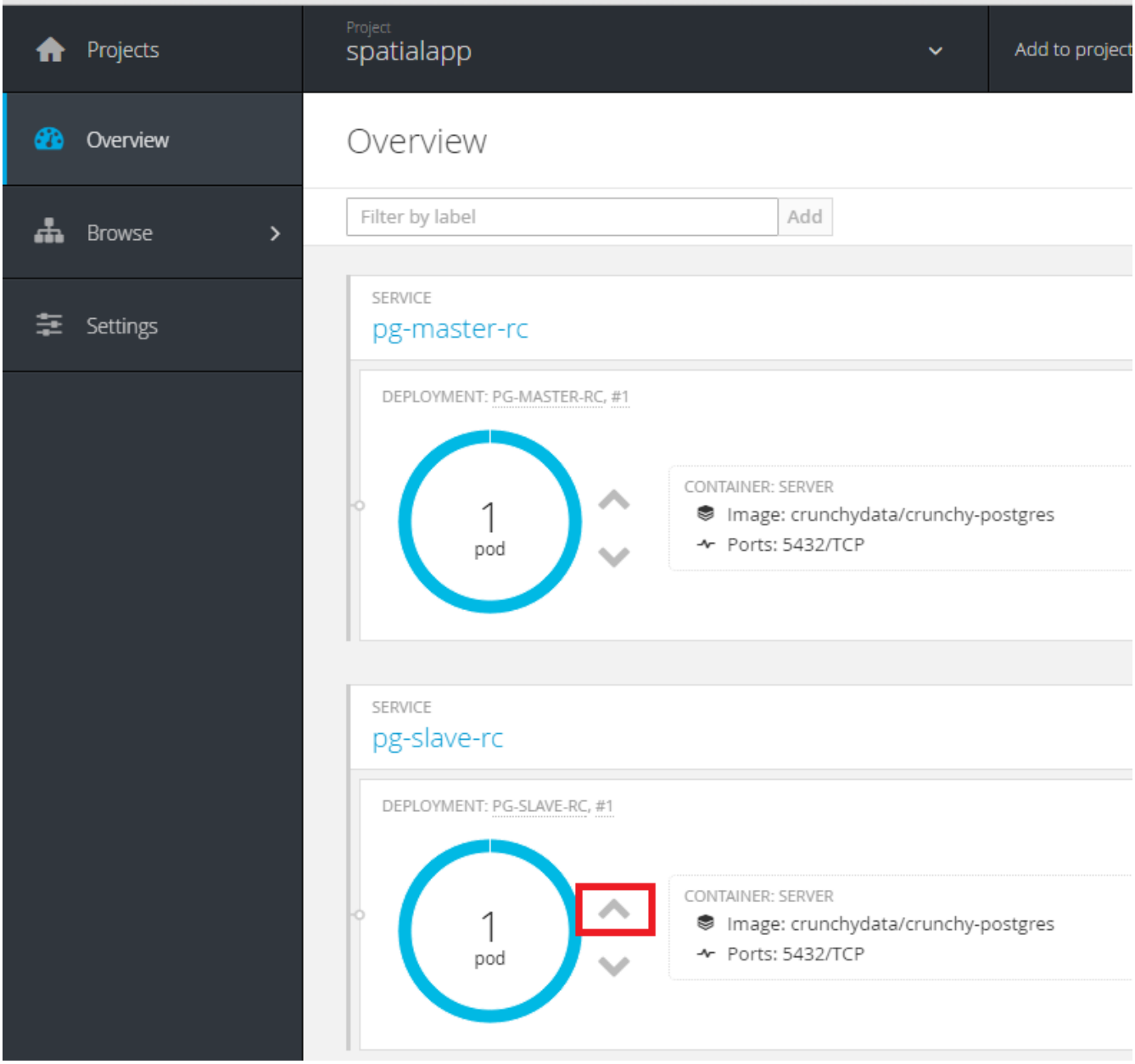
In that terminal go ahead and type the following commands:

```
sh-4.2$ psql userdb
psql (9.5.2)
Type "help" for help.

userdb=# select count(*) from parkpoints;
count
---
547
(1 row)
```

Do you REALIZE what just happened. We entered data into the Master DB and it was automatically replicated over to the slave DB and did 0 work to make sure that would happen.

Time for More Replication Magic.

Let's take this to even another level. In the web console, go back to the overview again and then click on the little up arrow next to the slave pods:



The number inside the circle will increment to 2 and then the blue circle will fill in the rest of the circle. You now have 2 replicas running. If you click on the circle again you will see the list of the two pods. If you click on the new pod and then do the commands above you will see that it has already been replicated to the new replica.

In the next section we will write an application to use the slaves and the replicas. Make sure you have cloned the v3simple-spatial repo. to the local machine.

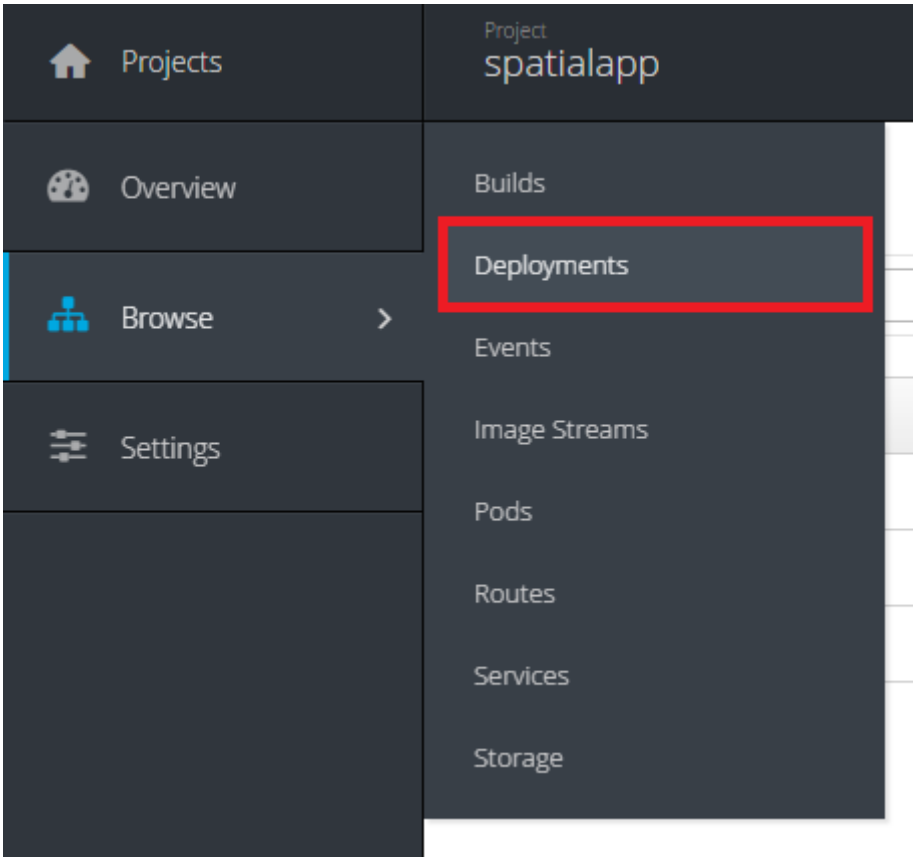
Simple Web Application

To exercise the database some more we are going to change our python web application to read and write from the PostGIS master and slave services. When using OpenShift the way to handle the connection paramters to the DB are handled through environment variables. Rather than hard coding in IP addresses or username/passwords we set environment variables that actually point to those values.

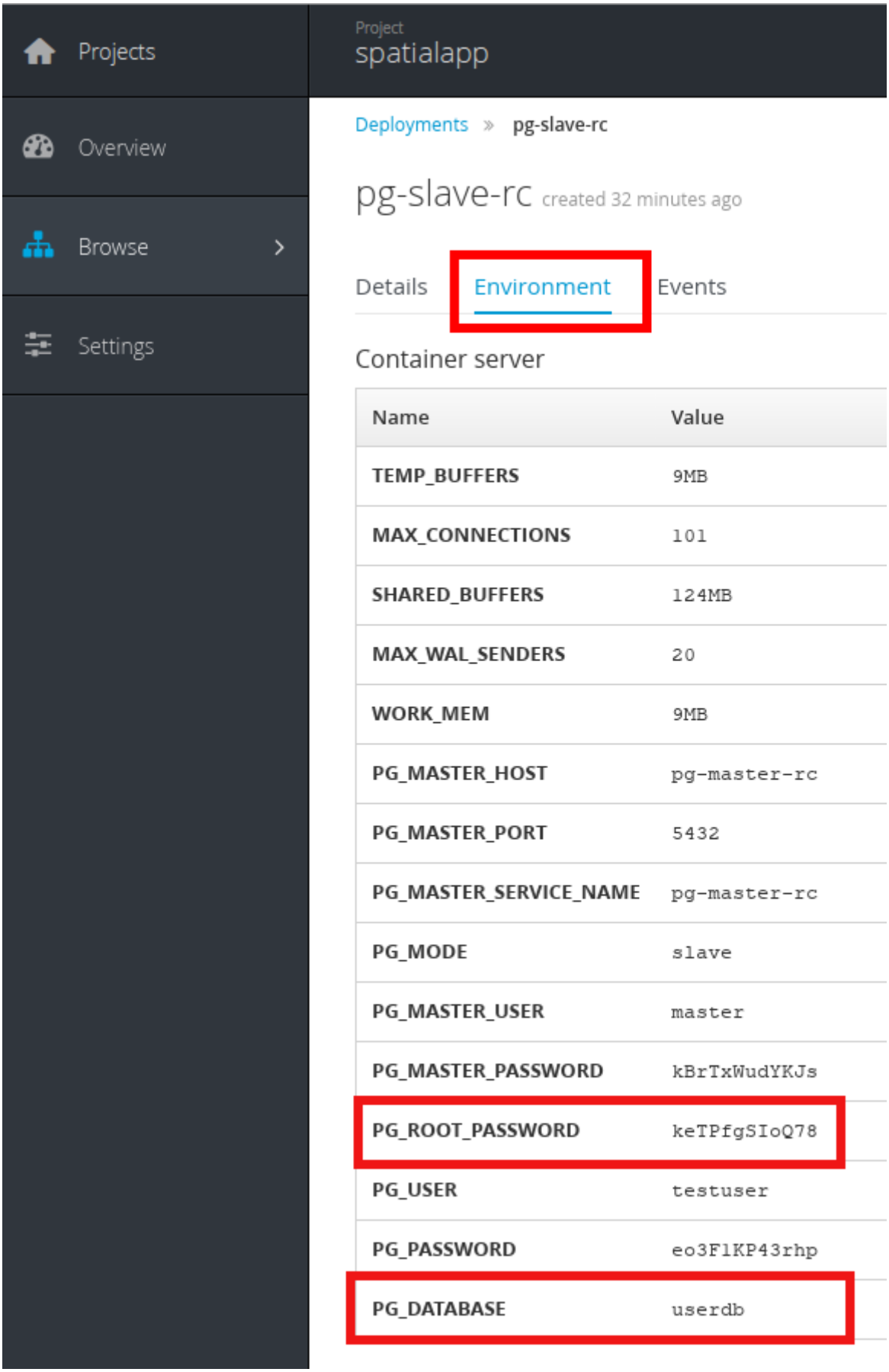
Some of these are set by the platform "automagically" - these are usually network type variables like hosts to IP and port mappings. The others, that we need to handle manually right now are database name, username, and password. At the end of this section I will explain how we could have avoided having to manually do this.

So as I explained above we got the network pieces for free. Let's add the env variables to the deployment configuration (dc) for the python code. We add it to the dc so it becomes available to all pods controlled by the dc. We will start with the read operations from the DB so we will use the slave pods for those operations.

In your browser go to the Browse → Deployments



From there click on the deployment for the *pg-slave-rc* then click on the environment tab. From there you should see all the environment variables defined on the dc. We are interested in 3 of the variables: PG_USER, PG_PASSWORD, and PG_DATABASE. You will need the names and values for both of these (please note that your values will be different). They are highlighted in red below:



Now that we have these we just have to do a simple command:

```
> oc set env dc/v3simple-spatial PG_DATABASE=userdb PG_USER=postgres
PG_ROOT_PASSWORD=WM4vPCCAGyt2
deploymentconfig "v3simple-spatial" updated
```

This will force a redeployment of the pods to get the new environment variables. We have to do this redeployment because remember, containers are immutable.

With that step done we can now modify our python code. I have the completed example as 2_app.py in the github repo. The steps we did to make this code change was

- Added the library to requirements.txt

```
psycopg2==2.6.1
```

- Added lines to the app.py to make the connection to the slave **service** with the proper credentials. Then on a GET request we return back some simple information from the database.

WARNING	This code is in NO WAY production type code. This code is trying to be as simple as possible so you learn the basic patterns. In a real app you would not load a DB connection on every request, you would check for exceptions, and you would have more optimized queries.

To make the code change there are two ways to proceed:

1. Either type or copy the code changes you see in 2_app.py into app.py
2. Go ahead and rename app.py to 1_app.py and then rename 2_app.py to app.py

After either of those steps go ahead and do the following commands in the v3simple-spatial directory:

```
> git commit -am "new code"
> git push origin master
```

Then back in your web console you can fire off another build like you did in the last exercise. After the build and deploy is finished when you go to http://v3simple-spatial-spatialapp.apps.10.2.2.2.xip.io/db the first 10 entries in the table.

Writing to Master

The great part of what we have set up is we can isolate our writes to master and our reads from the replica - which is why people usually set up replicas in the first place.

We have already set all the environment variables we need so really it is just editing the code. The only change we need to make to the new code is to have the connection be to the master rather than the replica. Other than that the connection parameters stay the same. In a more production

ready app you would probably use two different Postgresql accounts, one for reading and one for writing, which would require new environment variables.

I added code to randomly generate a name and the coords for a new point whenever you HTTP POST to the /db URL. Again this is really hacky code for a workshop - not production code. I will talk you through the code in class.

Remember, to make the code change there are two ways to proceed:

1. Either type or copy the code changes you see in 3_app.py into app.py
2. Go ahead and rename app.py to 2_app.py and then rename 3_app.py to app.py

Don't forget to do the git commands I gave you above and then fire off another build.

Finally, to hit this URL you can either install a plugin for your browser or you can use cURL. By default browsers do an HTTP GET but we need a POST. There are plenty of plugins for Chrome and Firefox to help you do a HTTP Post - most of them have the word REST in them. Here is cURL syntax that will exercise the end point:

```
# -d says to do a POST and we leave the payload blank
curl 'http://v3simple-spatial-spatialapp.apps.10.2.2.2.xip.io/db' -d ''

# if you want to look at the output in a nicer format you can save it to HTML
curl 'http://v3simple-spatial-spatialapp.apps.10.2.2.2.xip.io/db' -d ''  > index.html
```

If you use a browser plugin the URL stays the same and you just tell the plugin to use a POST.

The response will be the last 10 entries in the DB - which will include your latest entry. You can go ahead and POST several items and watch the new entries show up.

Advanced Usage - DB Metrics

In this section we are going to use some Docker containers built by CrunchyDB to get metrics on our databases. They have built containers to run Promethues and Grafana and then given us the templates to hook these up to our Postgresql Servers.

We already added the pieces to our Postgresql pods when we use the template during the earlier exercises. Now we need to setup the pieces to collect and display those metrics.

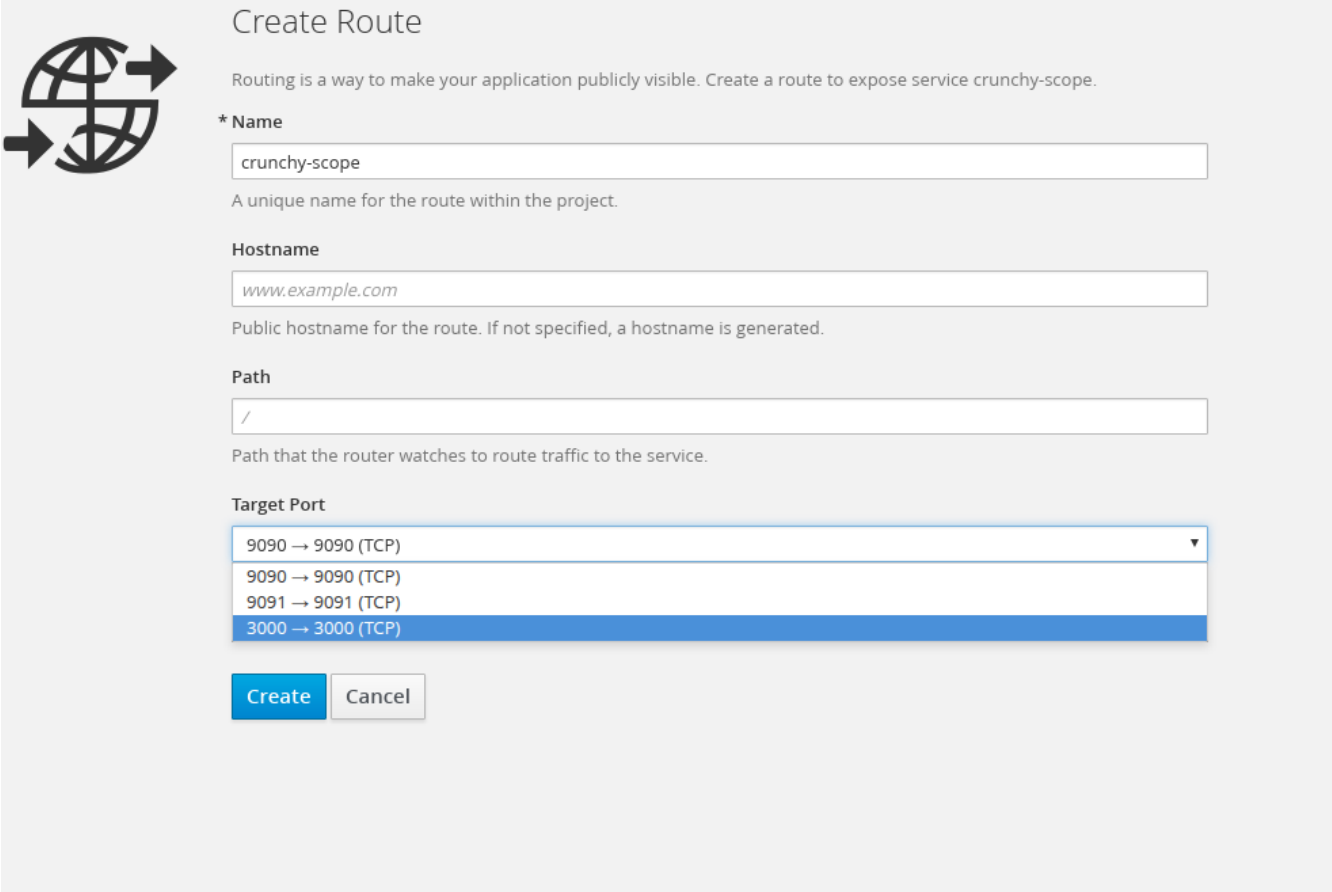
Adding the collection and display pieces

Go back into the crunchy-containers repo on your local machine and go into *examples/openshift/scope* then execture this command:

```
> oc new-app scope.json
```

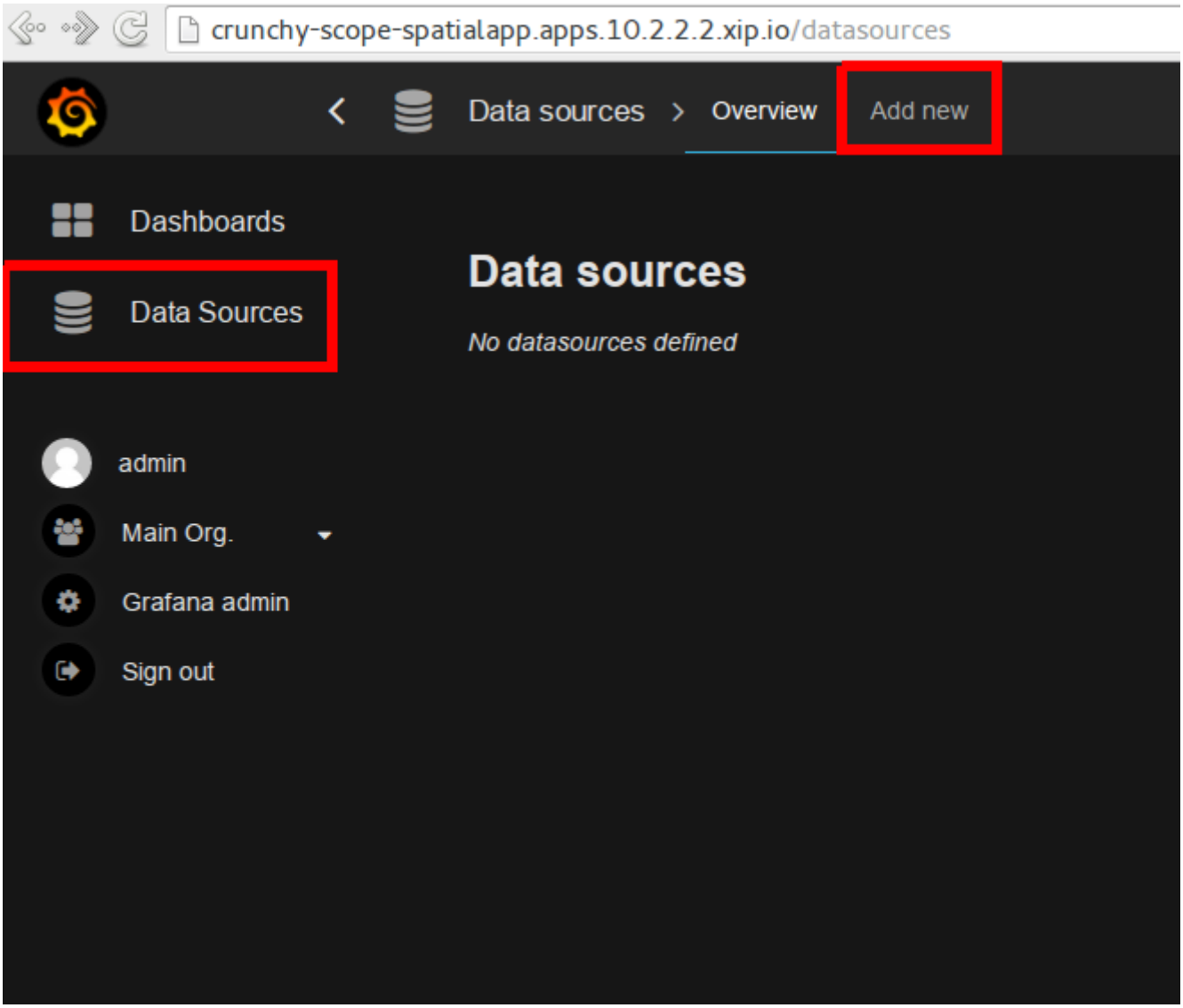
Now we need to expose the web port for these containers with a url. The Grafana pod is serviing up content over port 3000 and that is the one we want to expose.

Back in the web console you will now see the crunchy-scope service. Go ahead and click on the "create route" link on the right side of the service. This time before you click create make sure to change the port on the last dialog to 3000 -→ 3000.



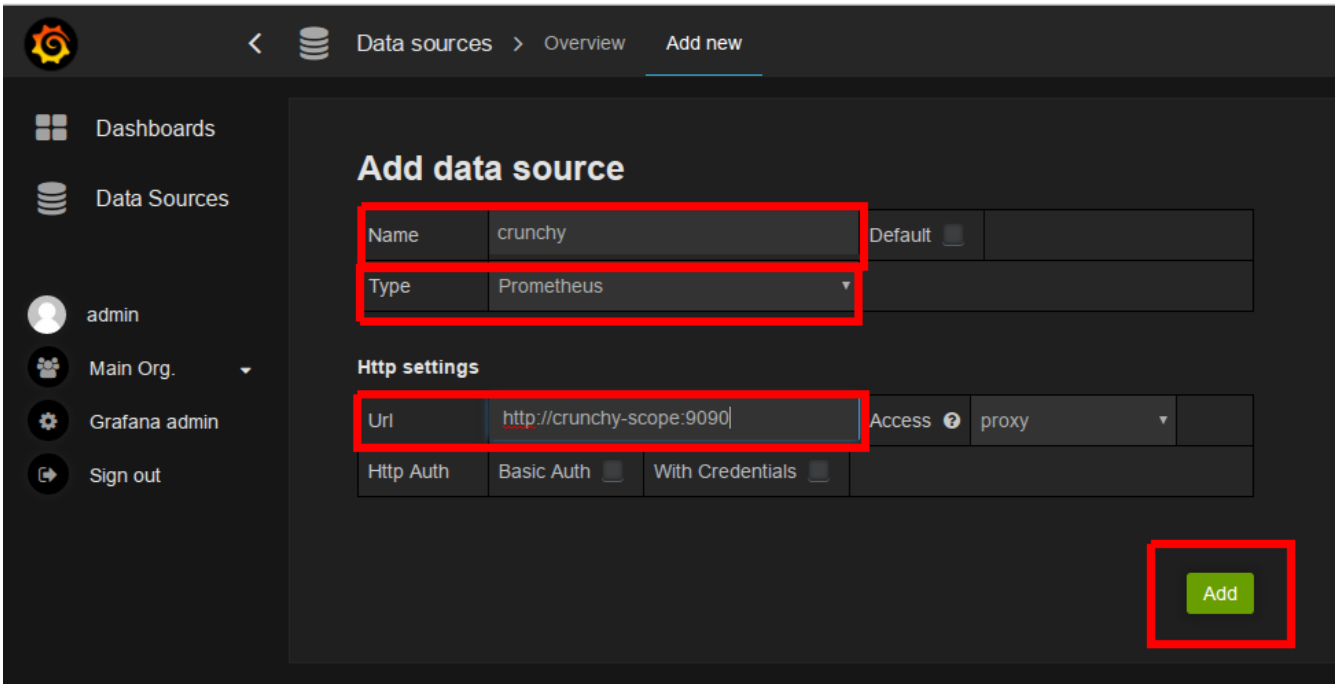
If you click on that link you should now see the home page for Grafana with a login prompt. Use admin for the username and the password.

Then there will be a with a lot of boxes and dialogs but before we do anything we have to add a Data Source. So go ahead and click on "Data Sources" and then "add new".



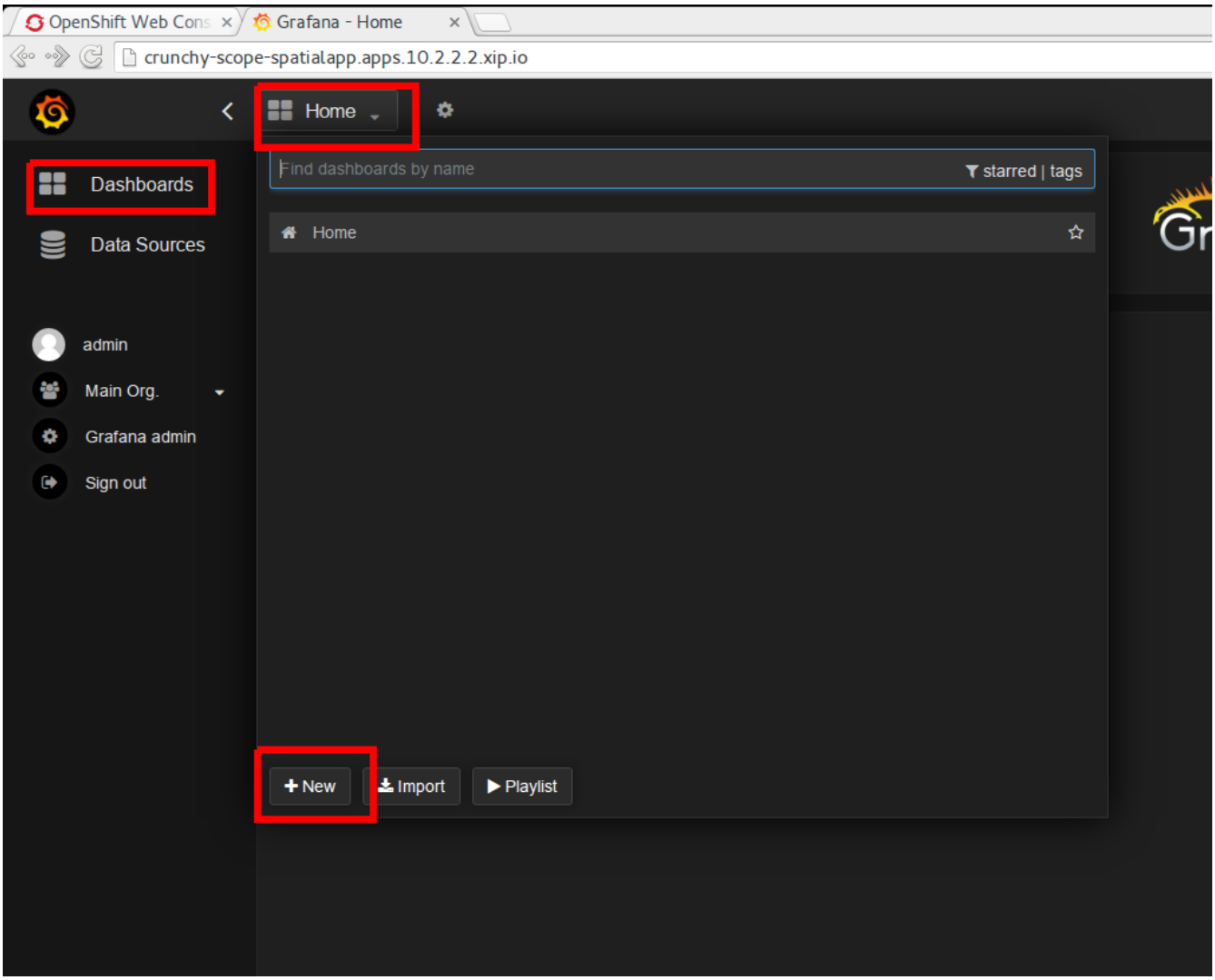
Then you get a much simpler box - go ahead and fill it out with the values you see in the boxes. In case you can't read the picture, the fields are:

- Name: crunchy
- Type: prometheus
- Url: http://crunchy-scope:9090



When that is all filled in go ahead and click the green "add" button in the bottom right corner. Then click the green "save" button.

Now that we have a data source we can start making graphs. Click on Dashboards in the top right, then click on home on the top and then "new" button on the bottom.



Now, very unintuitively you need to click on that little green bar on the top left that appears on the next screen. Follow the flow all the way down until you get to add a grah.

On the next (not so) intuitive page you need to first go to the bottom right and change the datasource to crunchy. Then you click in the Metric box and will pop-up a list of the fields you can query on. Not being a postgresql admin, I am not exactly sure which is the best to pick to show activity. We will also need to exercise the DB so go ahead and do some queries, hit the web page, anything to exercise the DB. As we go on we should see the metrics move.

And with that we have finished adding metrics to our database.

Here are some basic commands I found useful

How to build the documentation

```
asciidoctor -S unsafe ~/git/workshops/index.asciidoc
```

How to delete the application pieces but not the DB pieces

```
oc delete is,dc,svc,bc v3simple-spatial
```

How to do a web request with cURL

```
curl 'http://v3simple-spatial-spatialapp.apps.10.2.2.2.xip.io/db' -d ''
```

How to insert a value into the table

```
Insert into parkpoints (name, the_geom) VALUES ('ASteve', ST_GeomFromText('POINT(-
85.7302 37.5332)', 4326));
```