Cloud Foundry Application Runtime

Lab 12 – Getting Started with Cloud Foundry

Cloud Foundry Application Runtime (CFAR), often referred to as Cloud Foundry, is an open source, multi cloud application platform as a service (PaaS) governed by the Cloud Foundry Foundation. The software was originally developed by VMware and then transferred to Pivotal Software, a joint venture by EMC, VMware and General Electric with key follow on investments from Ford and Microsoft.

Cloud Foundry is used in continuous delivery environments, supplying support for the full application development lifecycle, from testing to deployment. Cloud Foundry's container-based architecture runs apps over a variety of cloud service providers.

Users have access to one or more spaces, which typically correspond to a lifecycle stage. For example, an application ready for QA testing might be pushed (deployed) to its project's QA space. Different users can be restricted to different spaces with different access permissions in each.

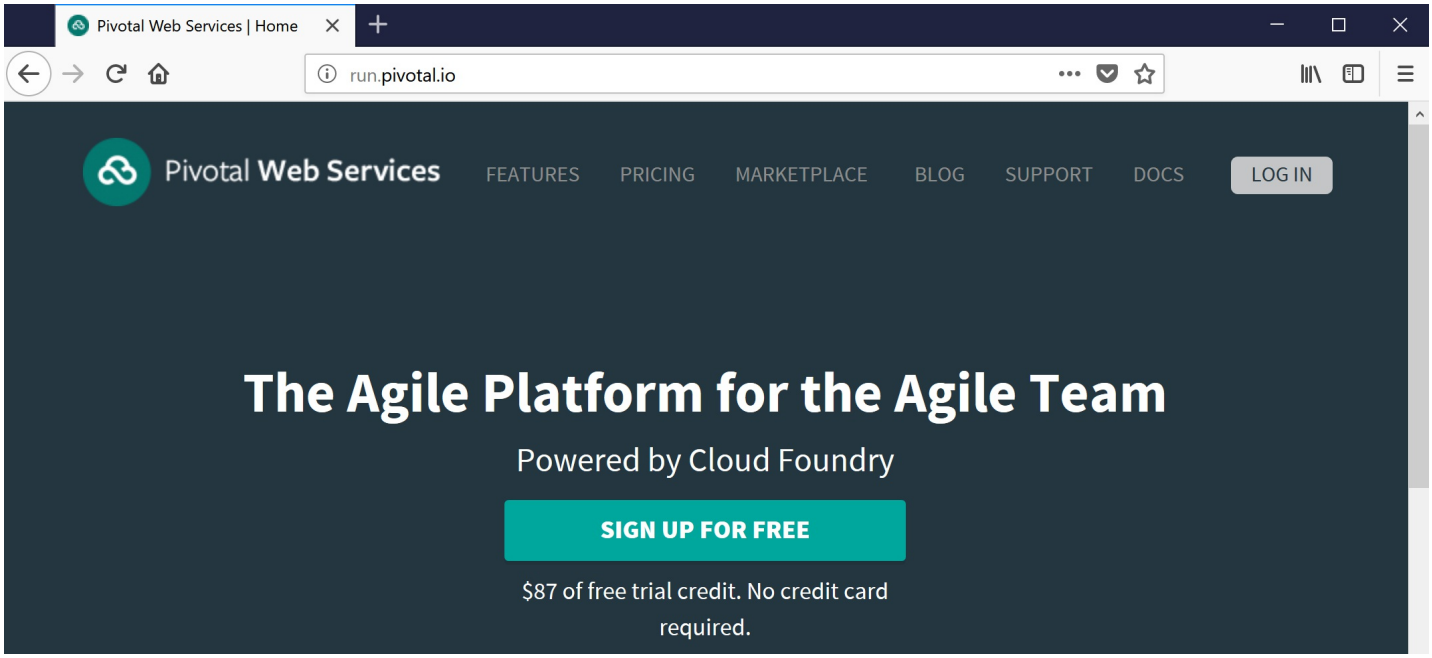
In this lab we will create a Cloud Foundry "space" and deploy a simple application to it. We will use the Pivotal Web Services (PWS) Cloud Foundry as a service platform to run our application.

1. Creating a PWS Account

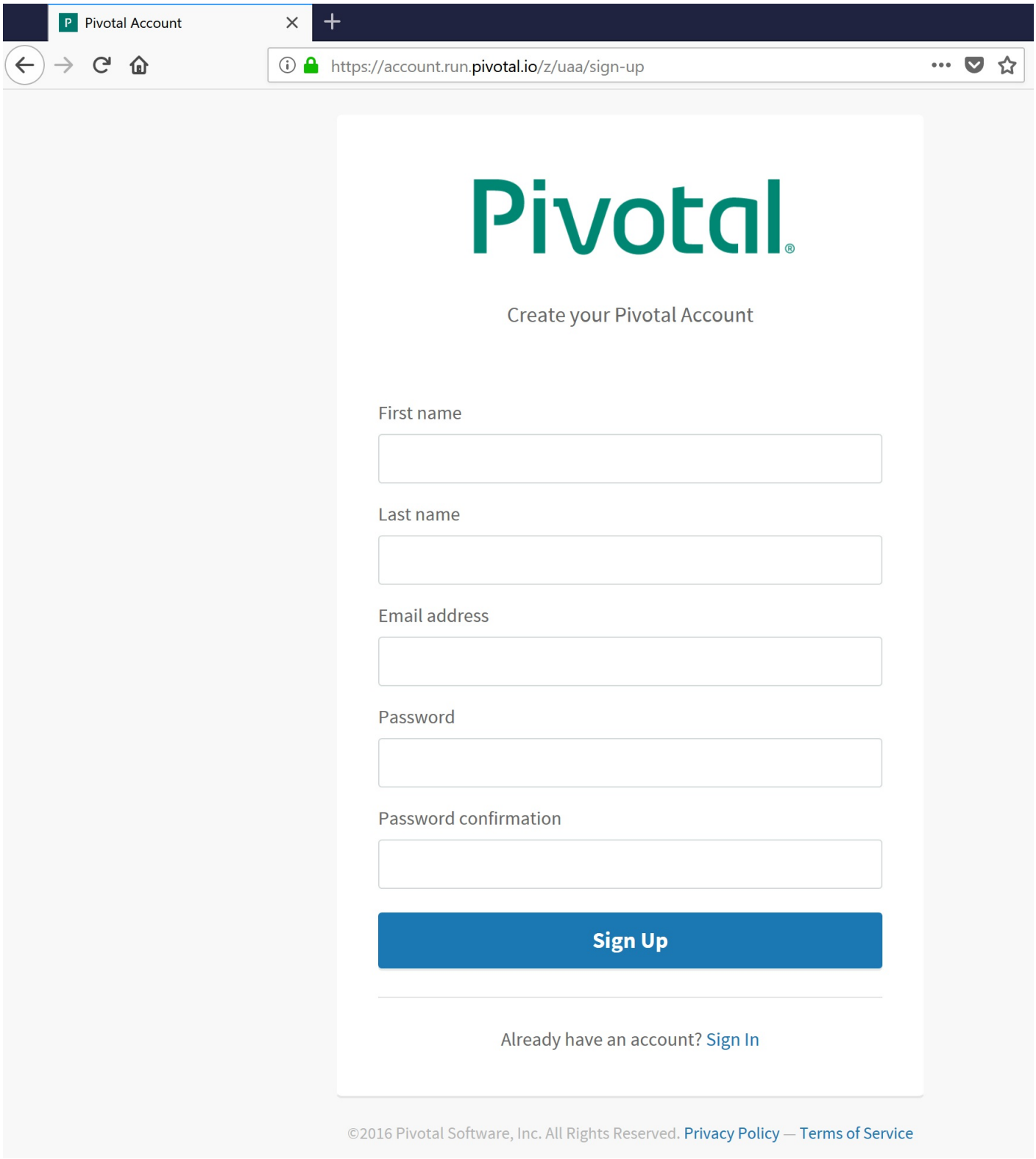
Pivotal Web Services (PWS) is a Cloud Foundry as a Service solution offered by Pivotal Software Inc. While PWS is a commercial service, you can sign up for a free PWS account at http://run.pivotal.io/

Each customer is eligible for one trial organization. A trial org has 2GB of memory and $87 of Trial credits good for one year. Your trial will end when you either use up all your Trial credits or a year has passed. Trial credits can be used towards app and task usage. Paid service plans are not available to trial orgs. You can add your credit card to your trial org at anytime which will give you access to paid service plans and 25GB of memory. Your remaining trial credits will be rolled into your paid plan.

Use a browser to open the http://run.pivotal.io/ IRI and click "SIGN UP FOR FREE":



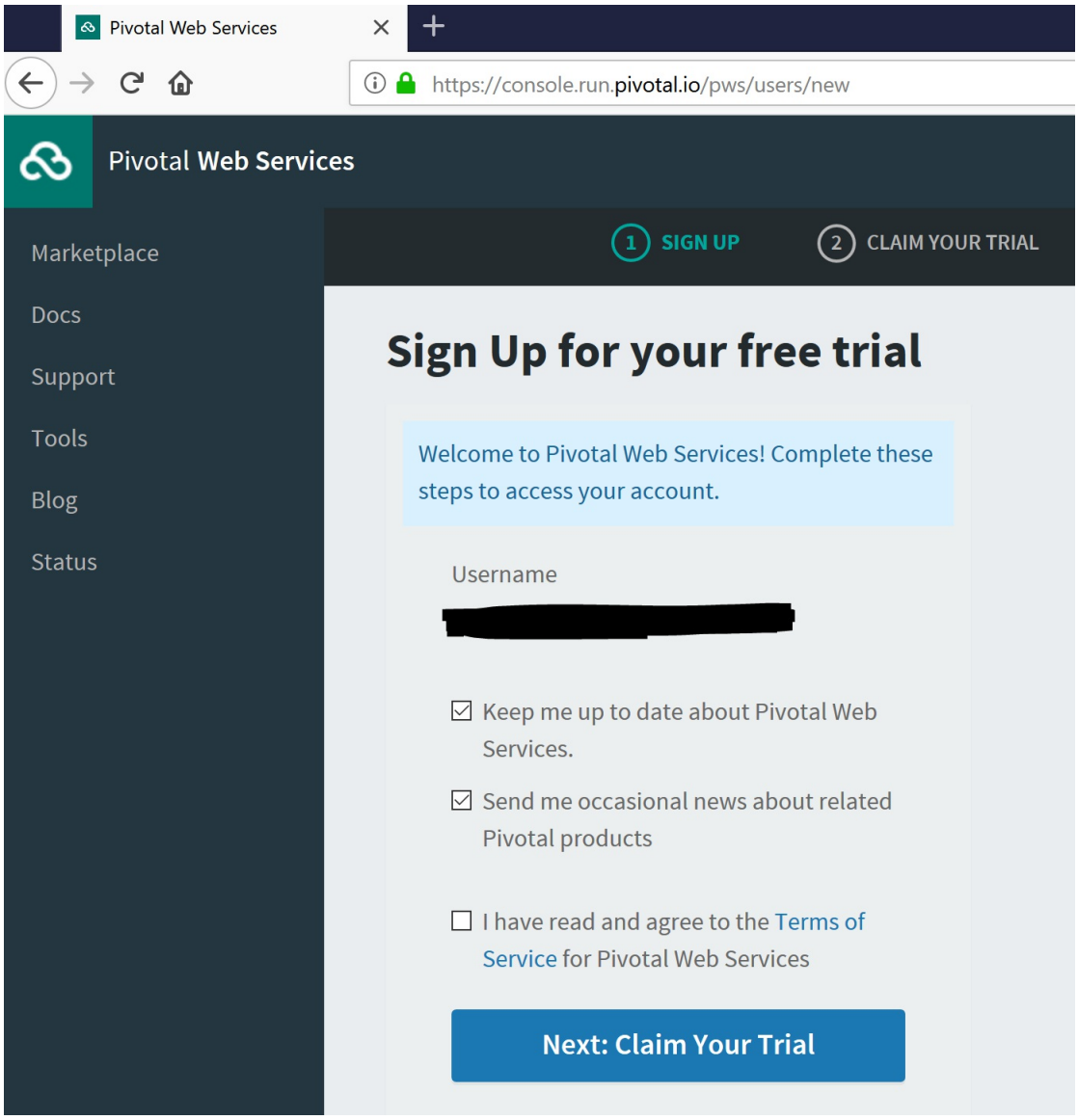
Fill out the sign up form with a valid email address and click "Sign Up:"



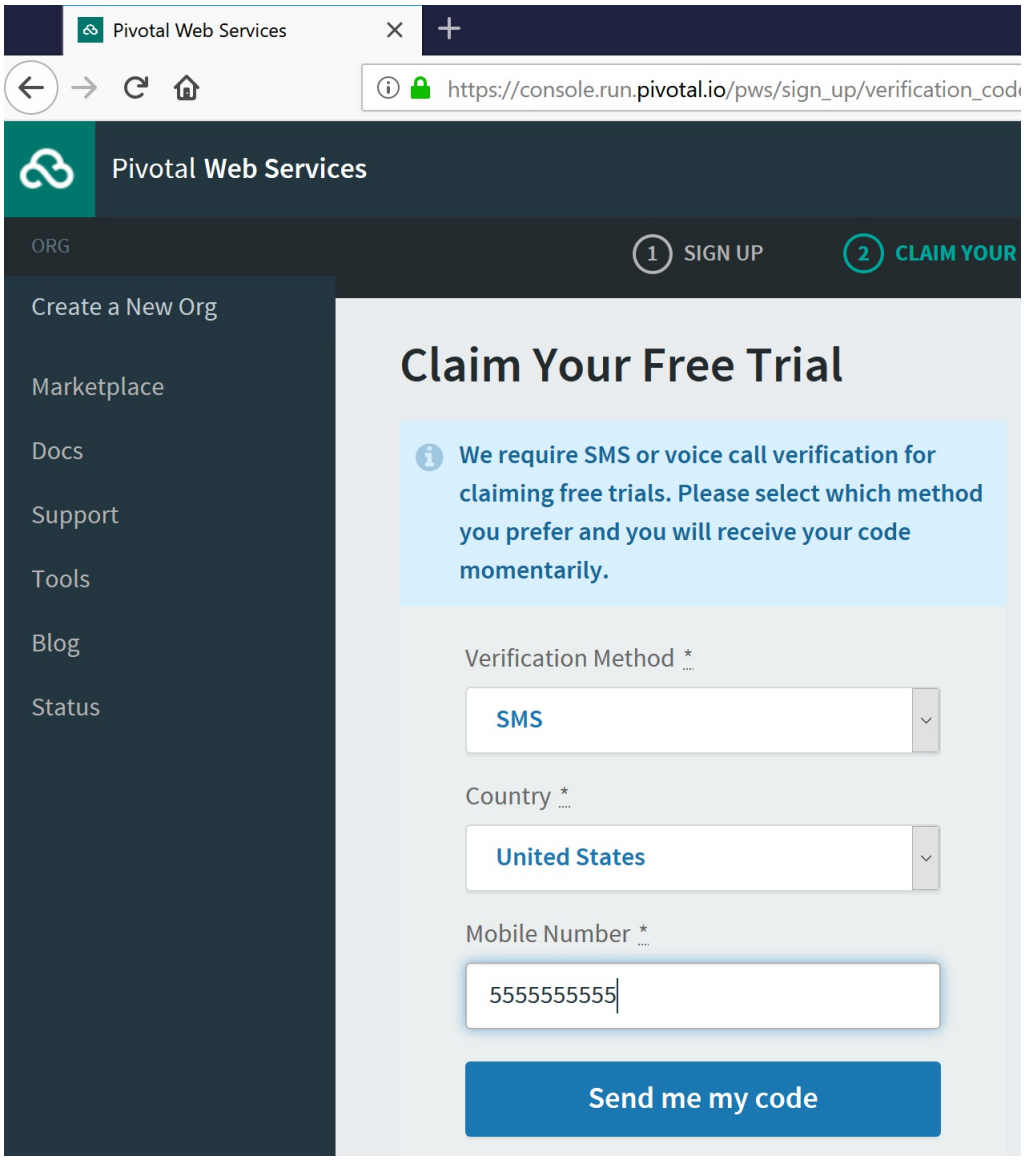
You should receive a sign up link in the inbox for the email address provided within seconds.

```
Your email address was used to create an account with Pivotal and requires verification.

Verify your email address

If you did not sign up for this account, just ignore this message.
```

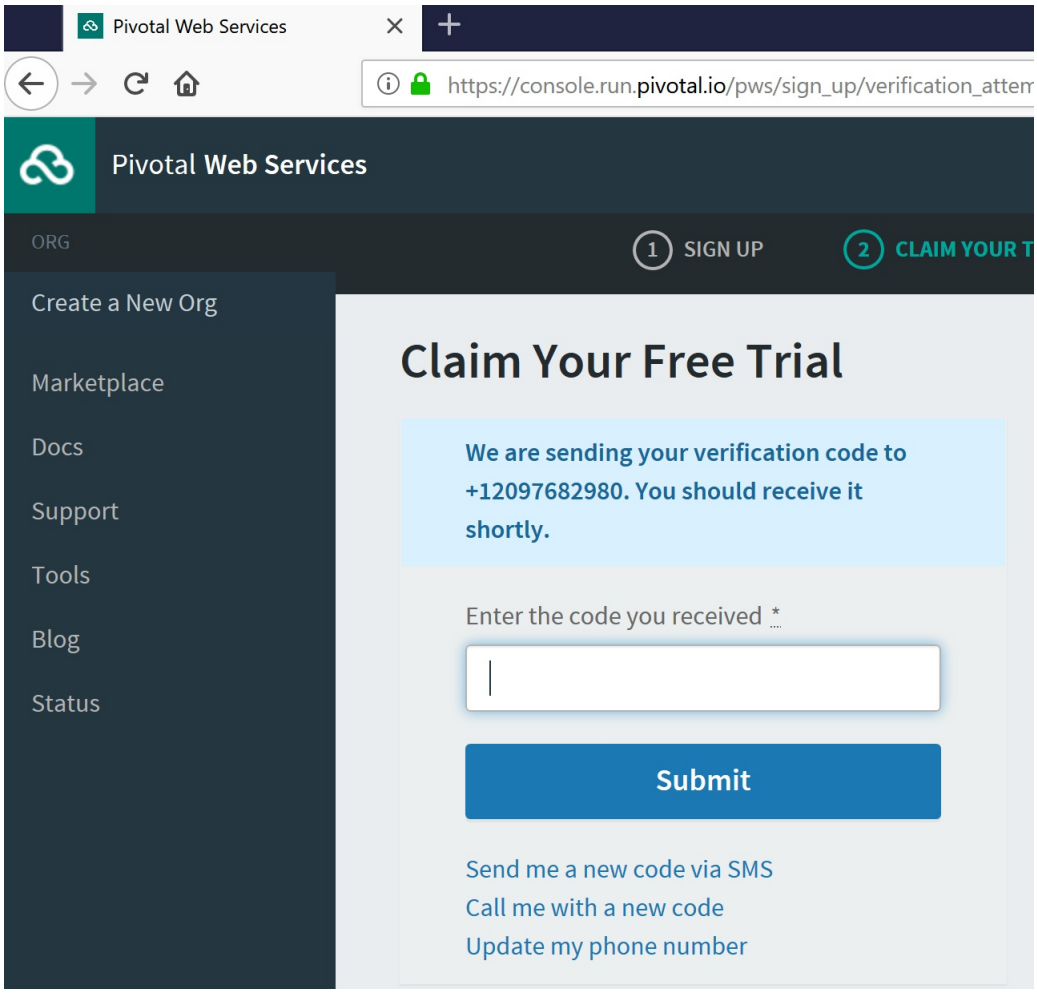
Click the verify email address link to return to the PWS web site. This should take you to the free trial claim page (if not just return to http://run.pivotal.io/ and click "sign in", then use the email and password you provided to create your account to login):



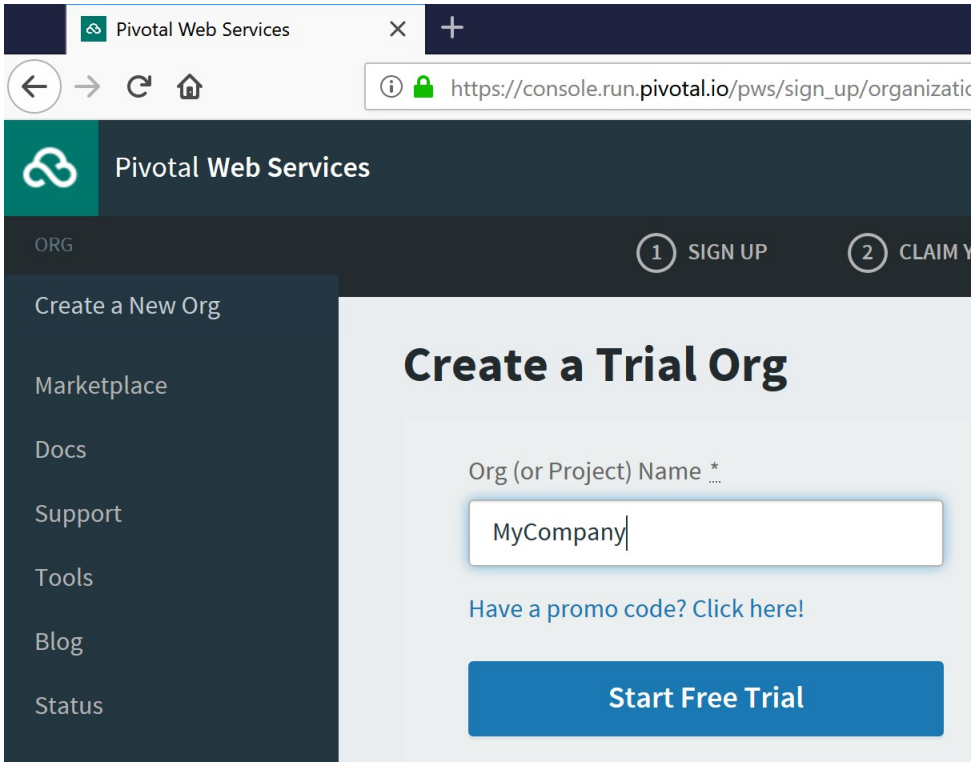
Check the "I have read and agree to the Terms of services ..." (if you do) and then click "Next: Claim Your Trial". Finally you will need to verify your trial with a cell phone number:



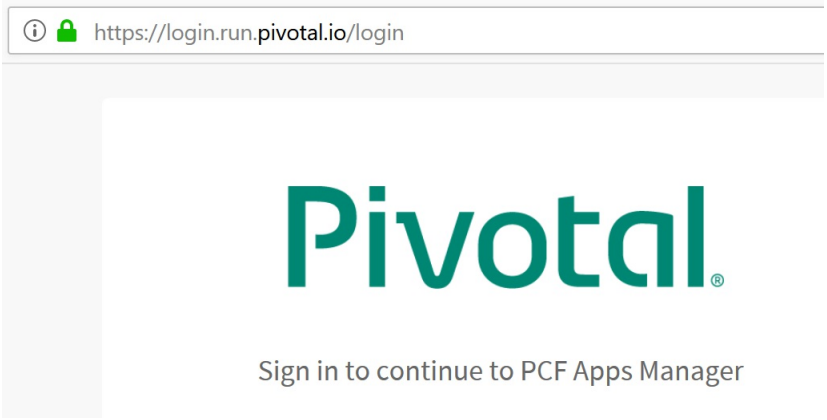
Enter a mobile phone number and PWS will text you a code. Enter the code in the next screen:



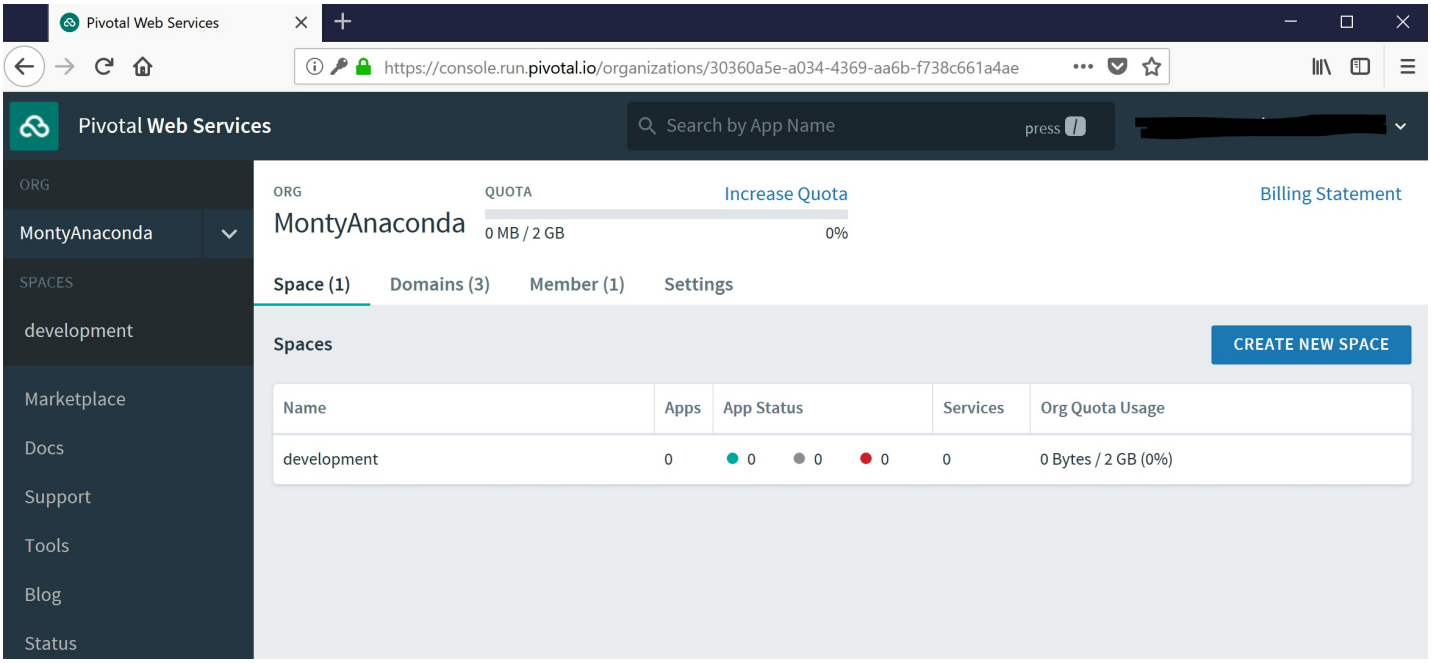
Now "Submit" the code to access the Organization setup screen:



Enter a name for the fictional (or real) organization that will host all of your deployed applications and click "Start Free Trial".
When the login screen comes up, click the email address you provided to login with it:

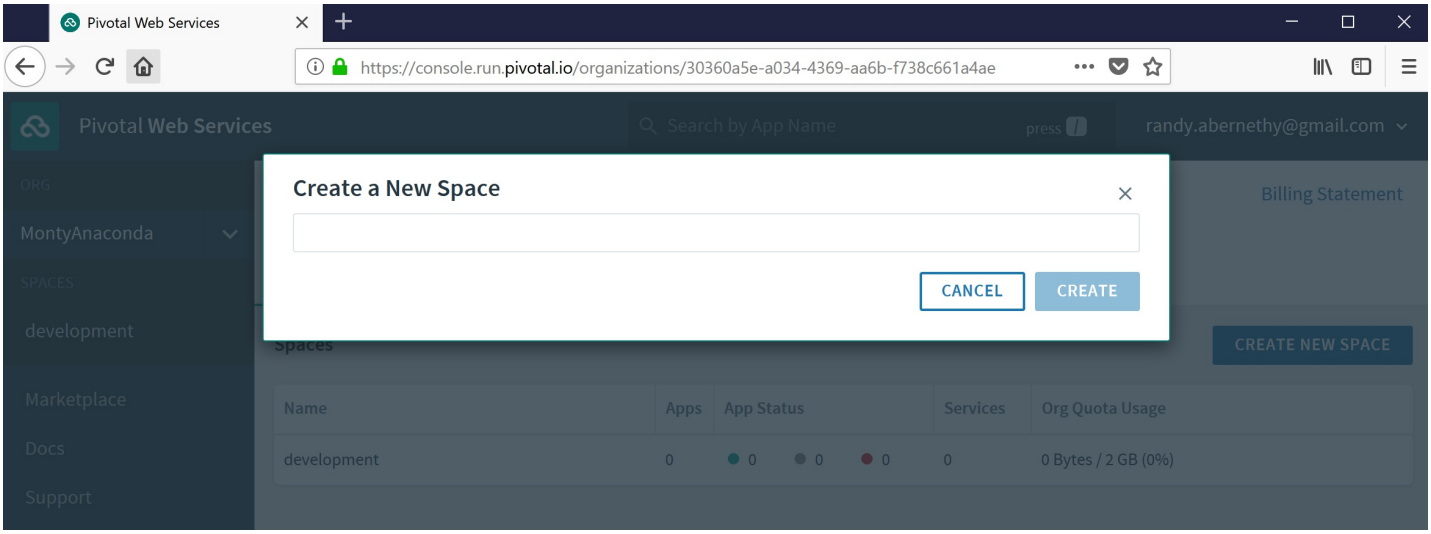


Now enter your password and click "Sign in". This should drop you into the main PWS screen:



The initial configuration includes a "development" space.

Click the "Create New Space" button to add another space:



Add a new space called UAT by entering "UAT" into the dialog and clicking "CREATE". You should now see both the "development" space and the "UAT" space in the Spaces table.

The left hand navigation pane allows you to easily navigate through organizations and spaces. The organization pull down allows you to create

additional organizations if you need to.

Click the "Domains" tab under your organization name. You should see three shared domains listed:

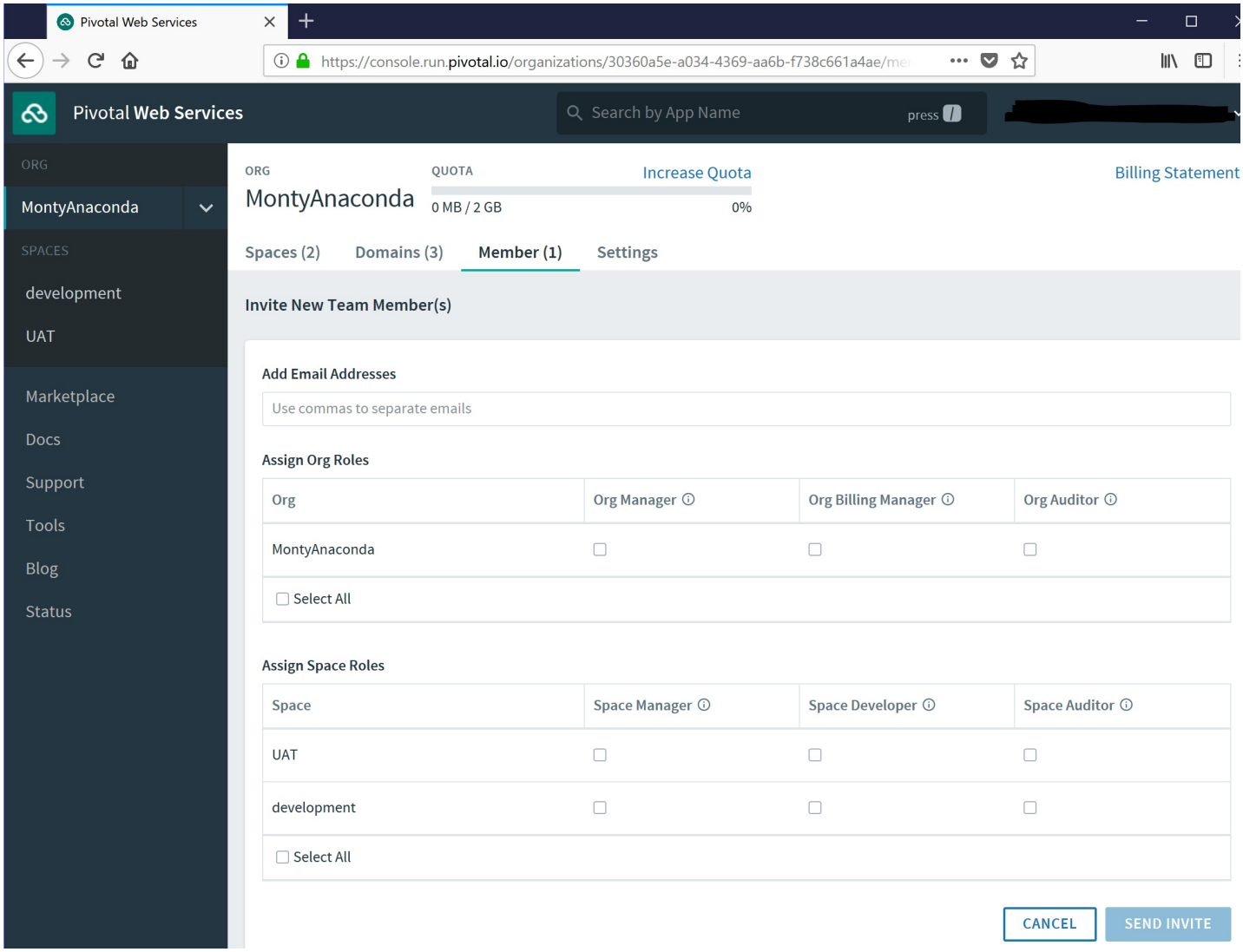
- apps.internal shared
- cf-tcpapps.io shared
- cfapps.io shared

You can add your own domain to the list of DNS suffixes available to your applications or use one of the shared domains provided by Pivotal.

Click on the "Members" tab next to the domain tab. You should see yourself listed with the following roles:

- Org Manager - can invite user and manage user roles in all spaces
- Org Billing Manager - can edit billing and payment information
- Org Auditor - read-only access to org information and reports

As you can see you are all powerful. Click the "INVITE NEW MEMBERS" button:

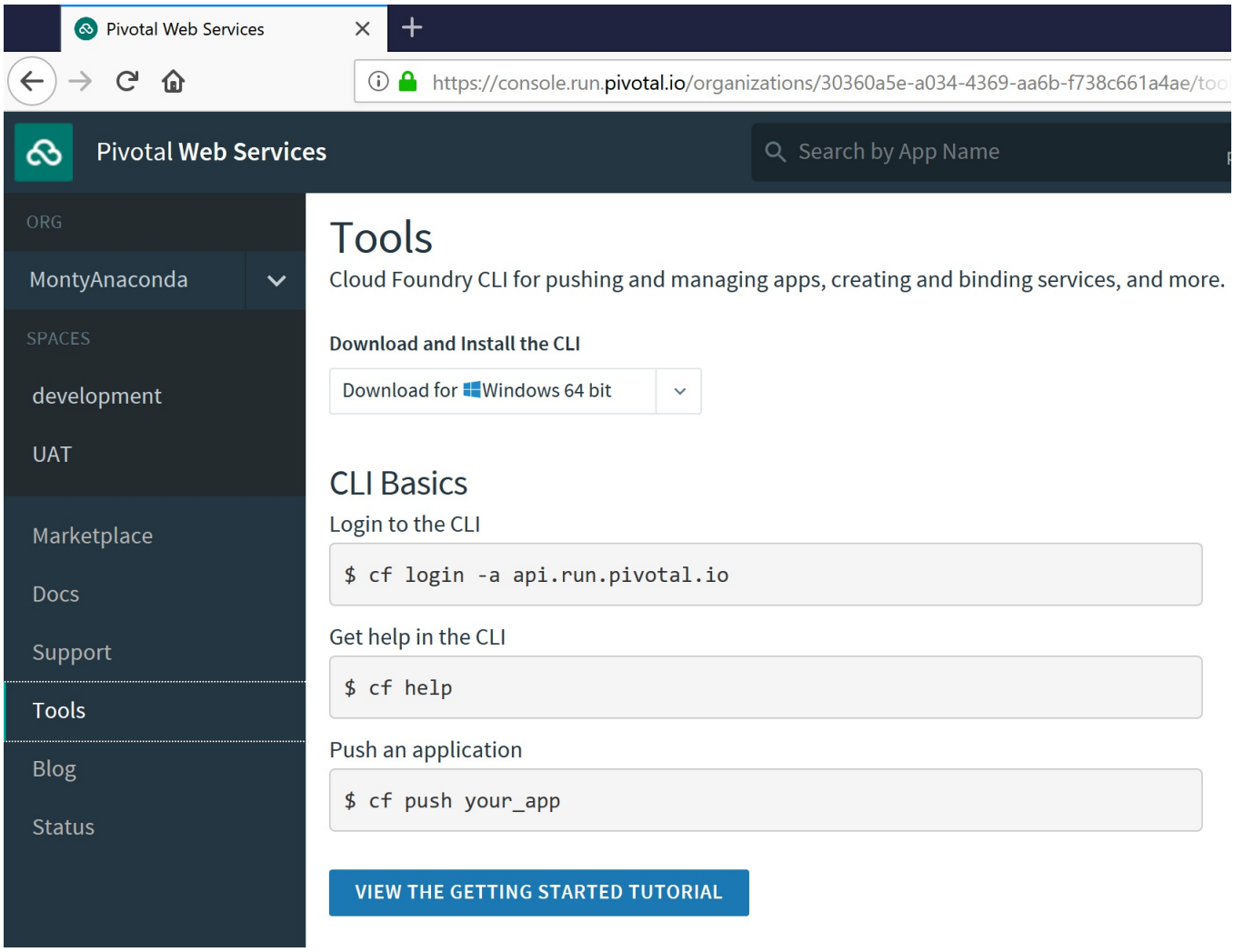


New members invited to the Organization can be given any of your three Organization roles or limited to a specific space with these permissions:

- Space Manager - can invite users and manage user roles in in this space only
- Space Developer - can create and manage all application features
- Space Auditor - read-only access to space information and reports

Great, you now have a full features Cloud Foundry PaaS environment setup and ready to use.

Next we need to setup the CF CLI tool. Click the "Tools" item in the left hand navigation pane:



This tells us how to login from the command line, how to get help, and how to push an application to Cloud Foundry, however we need to install the cf CLI first!

2. Installing the Cloud Foundry CLI

Developers make heavy use of the Cloud Foundry cf Command Line tool. The cf utility provides many options, but perhaps the most important command is "cf push", which deploys an application from your desktop to Cloud Foundry. The cf push command accepts arguments to specify the name of the application, where to load it from and the URL that should be used to access it.

Let's install the cf Command Line Interface (CLI) on your lab system. First add the Cloud Foundry repository key to the apt package manager's key list:

```
user@ubuntu:~$ wget -q -O - https://packages.cloudfoundry.org/debian/cli.cloudfoundry.org.key | sudo apt-key add -

OK

user@ubuntu:~$
```

Now add the Cloud Foundry package manager URL to the apt sources:

```
user@ubuntu:~$ echo "deb http://packages.cloudfoundry.org/debian stable main" | sudo tee
/etc/apt/sources.list.d/cloudfoundry-cli.list

deb http://packages.cloudfoundry.org/debian stable main

user@ubuntu:~$
```

Now update the package indexes to pickup the new Cloud Foundry packages:

```
user@ubuntu:~$ sudo apt-get update

Hit:1 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
```

```
Get:6 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [465 kB]
Ign:3 https://cf-cli-debian-repo.s3.amazonaws.com stable InRelease
Get:8 https://cf-cli-debian-repo.s3.amazonaws.com stable Release [1,797 B]
Get:9 https://download.docker.com/linux/ubuntu xenial InRelease [65.8 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [744 kB]
Get:11 https://cf-cli-debian-repo.s3.amazonaws.com stable Release.gpg [819 B]
Get:12 https://cf-cli-debian-repo.s3.amazonaws.com stable/main amd64 Packages [7,172 B]
Get:13 https://cf-cli-debian-repo.s3.amazonaws.com stable/main i386 Packages [7,175 B]
Get:14 http://security.ubuntu.com/ubuntu xenial-security/main i386 Packages [419 kB]
Hit:5 https://packages.cloud.google.com/apt kubernetes-xenial InRelease
Get:15 http://us.archive.ubuntu.com/ubuntu xenial-updates/main i386 Packages [690 kB]
Get:16 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [615 kB]
Get:17 http://us.archive.ubuntu.com/ubuntu xenial-updates/universe i386 Packages [570 kB]
Fetched 3,893 kB in 4s (948 kB/s)
Reading package lists... Done

user@ubuntu:~$
```

Now install the Cloud Foundry CLI package:

```
user@ubuntu:~$ sudo apt-get install cf-cli

...

user@ubuntu:~$
```

Test the cf tool by displaying the version:

```
user@ubuntu:~$ cf --version

cf version 6.35.2+88a03e995.2018-03-15

user@ubuntu:~$
```

Perfect cf CLI installed and working!

3. CLI Login and Space Operations

Login to your Cloud Foundry account and activate the UAT space:

```
user@ubuntu:~$ cf login -a api.run.pivotal.io

API endpoint: api.run.pivotal.io

Email> fred.flintstone@gmail.com

Password>
Authenticating...
OK

Targeted org MontyAnaconda

Select a space (or press enter to skip):
1. development
2. UAT

Space> 2
Targeted space UAT



API endpoint:   https://api.run.pivotal.io (API version: 2.106.0)
User:           fred.flintstone@gmail.com
Org:            MontyAnaconda
Space:          UAT

user@ubuntu:~$
```

You can always change spaces later or push to a specific space if you need to.

Try using the cf command to display the spaces in your org:

```
user@ubuntu:~$ cf spaces
```

```
Getting spaces in org MontyAnaconda as randy.abernethy@gmail.com...

name
development
UAT

user@ubuntu:~$
```

Now try listing details for the UAT space:

```
user@ubuntu:~$ cf space UAT

Getting info for space UAT in org MontyAnaconda as randy.abernethy@gmail.com...

name:                      UAT
org:                       MontyAnaconda
apps:
services:
isolation segment:
space quota:
running security groups:   credhub-internal-z2, credhub-internal-z3, dns, p-mysql, p.mysql, pcf-rabbitmq-
multitenant-prod, pcf-redis, public_networks, rabbitmq,
                           ssh-logging, udp80-logging
staging security groups:   credhub-internal-z2, credhub-internal-z3, dns, public_networks

user@ubuntu:~$
```

Note that the security groups configured by default give some indication of the services and features available in the space:

- credhub-internal-z2 and z3 - The Credentials Hub used by Cloud Foundary to manage platform credentials
- dns - DNS services
- p-mysql - MYSQL database service
- pcf-rabbitmq-multitenant-prod - Multitenant RabbitMQ service (inter org communication)
- pcf-redis - REDIS key value store
- public_networks - Internet access
- rabbitmq - RabbitMQ
- ssh-logging - Secure Shell login logging
- udp80-logging - QUIK protocol logging

The target command allows you to change your deployment (push) target. Get help on the target command:

```
user@ubuntu:~$ cf help target

NAME:
   target - Set or view the targeted org or space

USAGE:
   cf target [-o ORG] [-s SPACE]

ALIAS:
   t

OPTIONS:
   -o      Organization
   -s      Space

SEE ALSO:
   create-org, create-space, login, orgs, spaces

user@ubuntu:~$
```

Change your space target to development:

```
user@ubuntu:~$ cf target -s development

api endpoint:   https://api.run.pivotal.io
api version:    2.106.0
user:           randy.abernethy@gmail.com
org:            MontyAnaconda
space:          development

user@ubuntu:~$
```

Now use the target command (you can use the "t" alias for short) to display your deploy target:

```
user@ubuntu:~$ cf t

api endpoint:   https://api.run.pivotal.io
api version:    2.106.0
user:           randy.abernethy@gmail.com
org:            MontyAnaconda
space:          development

user@ubuntu:~$
```

4. Creating a service to Deploy

Now we're ready to deploy an application. We'll create a trivial NodeJS application to test Cloud Foundry.

To begin create a working directory for your application:

```
user@ubuntu:~$ mkdir cfar

user@ubuntu:~$ cd cfar

user@ubuntu:~/cfar$
```

Our JavaScript app will simply listen on a port and respond to GET operations on the "/status" route with the message "Server A". We'll use the popular Express library to construct our simple REST interface. Here's the code:

```
var express = require("express");
var http = require("http");
var app = express();

app.get("/status", function(req, res){
  return res.send('Server A\n');
});

http.createServer(app).listen(process.env.PORT, function() {
  console.log("Listening on port " + process.env.PORT);
});
```

Note that we can not hard code the listing port. Cloud Foundry may run multiple containers on the same host. To insure that each application listens on a separate port CF injects a PORT environment variable into each container with the PORT it expects the service to listen on. CF will then try to connect to that port to ensure that the application is health. In our code we set the listening port to `process.env.PORT`, which is the NodeJS way to retrieve an environment variable.

Enter the program into the file app.js:

```
user@ubuntu:~/cfar$ vim app.js

user@ubuntu:~/cfar$ cat app.js

var express = require("express");
var http = require("http");
var app = express();

app.get("/status", function(req, res){
  return res.send('Server A\n');
});

http.createServer(app).listen(process.env.PORT, function() {
  console.log("Listening on port " + process.env.PORT);
});

user@ubuntu:~/cfar$
```

5. Testing the service

if you do not have Docker installed, skip this step

Before we send the code to the cloud lets test it in a local container. Run a Node container, install express and then run the program:

```
user@ubuntu:~/cfar$ docker run -v "$PWD":/app -it --name cftest -e PORT=9090 node bash

root@e384945e2ec8:/# npm install express
```

```
npm WARN saveError ENOENT: no such file or directory, open '/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/package.json'
npm WARN !invalid#1 No description
npm WARN !invalid#1 No repository field.
npm WARN !invalid#1 No README data
npm WARN !invalid#1 No license field.
+ express@4.16.3
added 50 packages in 3.142s

root@e384945e2ec8:/# node app/app.js

Listening on port 9090
```

Note that the server is listening on PORT 9090 because we set the container's PORT environment variable to 9090 using the docker run -e switch.

While the app is running, open a new terminal to try curling the /status route. The service container was started with the name "cftest", so we can use the name to look up the container's IP address:

```
user@ubuntu:~$ docker container inspect cftest | grep -i ipaddr

            "SecondaryIPAddresses": null,
            "IPAddress": "172.17.0.2",
                    "IPAddress": "172.17.0.2",

user@ubuntu:~$
```

Now try curling the service with the service IP, port and "/status" route:

```
user@ubuntu:~$ curl 172.17.0.2:9090/status

Server A

user@ubuntu:~$
```

It works!.

Delete the test service container when you are finished experimenting:

```
user@ubuntu:~$ docker container stop cftest

cftest

user@ubuntu:~$ docker container rm cftest

cftest

user@ubuntu:~$
```

6. Pushing the Service to Cloud Foundry

In the step above we had to do several things to run our service.

1. Run a NodeJS container
2. Install Express
3. Optionally specify a Port to listen to using an environment variable
4. Run the program

We'll need to figure out how to tell Cloud Foundry about each of these requirements.

6.1. Tell CF to run a NodeJS container

Cloud Foundry has a built in buildpack for NodeJS. Detailed information for all of the CF BuildPacks is available on the web (you can find the NodeJS information here: https://docs.cloudfoundry.org/buildpacks/node/index.html). We'll use the NodeJS BuildPack to have CF deploy our application in a NodeJS container.

6.2. Tell CF to install Express

Node.JS applications generally define their dependencies in a package.json file. Using the package.json file we can tell CloudFoundry which version of Express we want installed and which version of NodeJS we want to use. We can only select NodeJS versions supported by the BuildPack. You can usually find the supported language versions in the BuildPack release notes. For Node they are here: https://github.com/cloudfoundry/nodejs-buildpack/releases

Here's the package.json file we'll use:

```
{
  "name": "cftest",
  "version": "0.0.1",
  "author": "FredFlintstone",
  "dependencies": {
    "express": "3.4.8"
  },
  "engines": {
    "node": "12.8.1"
  }
}
```

We give the application a name, version and author, then specify a supported Node version and a valid Express version.

Create the package.json:

```
user@ubuntu:~/cfar$ vim package.json

user@ubuntu:~/cfar$ cat package.json

{
  "name": "cftest",
  "version": "0.0.1",
  "author": "FredFlintstone",
  "dependencies": {
    "express": "3.4.8"
  },
  "engines": {
    "node": "12.8.1"
  }
}

user@ubuntu:~/cfar$
```

6.3. CF will select the PORT

The Cloud Foundry runtime will automatically select a port for our application and set the PORT environment variable.

6.4. Tell CF to run the app.js program

Now we can use cf push to deploy our application. We'll use the following parameters:

- cftest - the app name
- -i 1 - the number of instances of the service to launch
- -m 512M - the amount of memory to assign to the app
- -c "node app.js" - the command to use to launch the app
- -n cftest-1234 - the network name for the app

Try it but make one change to the example below, add your initials to the network name:

```
user@ubuntu:~/cfar$ cf push cftest -i 1 -m 512M -c "node app.js" -n cftest-1234

Pushing app cftest to org MontyAnaconda / space development as randy.abernethy@gmail.com...
Getting app info...
Updating app with these attributes...
  name:                cftest
  path:                /home/user/cfar
  buildpack:           nodejs
  command:             node app.js
  disk quota:          1G
  health check type:   port
  instances:           1
  memory:              512M
  stack:               cflinuxfs2
  routes:
    cftest-1234.cfapps.io

Updating app cftest...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
Uploading files...
 586 B / 586 B [================================================] 100.00% 1s
```

```
Waiting for API to complete processing files...

Stopping app...

Staging app and tracing logs...
   Downloading binary_buildpack...
   Downloading staticfile_buildpack...
   Downloading java_buildpack...
   Downloading dotnet_core_buildpack_beta...
   Downloading ruby_buildpack...
   Downloaded staticfile_buildpack
   Downloading nodejs_buildpack...
   Downloaded binary_buildpack
   Downloading go_buildpack...
   Downloaded java_buildpack
   Downloading python_buildpack...
   Downloaded ruby_buildpack
   Downloading php_buildpack...
   Downloaded nodejs_buildpack
   Downloading dotnet_core_buildpack...
   Downloaded go_buildpack
   Downloading hwc_buildpack...
   Downloaded hwc_buildpack
   Downloaded dotnet_core_buildpack_beta
   Downloaded python_buildpack
   Downloaded php_buildpack
   Downloaded dotnet_core_buildpack
   Cell 427f4537-d113-4050-8f1f-eb1875759ae8 creating container for instance b68559a4-73cb-4f68-b981-3b4c9441f7ae
   Cell 427f4537-d113-4050-8f1f-eb1875759ae8 successfully created container for instance b68559a4-73cb-4f68-b981-
3b4c9441f7ae
   Downloading build artifacts cache...
   Downloading app package...
   Downloading build artifacts cache failed
   Downloaded app package (586B)
   -----> Nodejs Buildpack version 1.6.20
   -----> Installing binaries
          engines.node (package.json): 4.8.6
          engines.npm (package.json): unspecified (use default)
   -----> Installing node 4.8.6
          Copy [/tmp/buildpacks/72bd2041bedd176395447e2e13206b5a/dependencies/b91283ce56875c88c40f8979a06f4744/node-
4.8.6-linux-x64-c8323162.tgz]
          **WARNING** A newer version of node is available in this buildpack. Please adjust your app to use version
4.8.7 instead of version 4.8.6 as soon as possible. Old versions of node are only provided to assist in migrating to
newer versions.
          **WARNING** node 4.x will no longer be available in new buildpacks released after 2018-04-01.
          See: https://github.com/nodejs/LTS
          Using default npm version: 2.15.11
   -----> Installing yarn 1.5.1
          Copy [/tmp/buildpacks/72bd2041bedd176395447e2e13206b5a/dependencies/dcc767a2459b6c3a81150e93c9e7973a/yarn-
v1.5.1-cd316572.tar.gz]
          Installed yarn 1.5.1
   -----> Creating runtime environment
          PRO TIP: It is recommended to vendor the application's Node.js dependencies
          Visit http://docs.cloudfoundry.org/buildpacks/node/index.html#vendoring
          NODE_ENV=production
          NODE_HOME=/tmp/contents141953331/deps/0/node
          NODE_MODULES_CACHE=true
          NODE_VERBOSE=false
          NPM_CONFIG_LOGLEVEL=error
          NPM_CONFIG_PRODUCTION=true
   -----> Restoring cache
          Skipping cache restore (no previous cache)
   -----> Building dependencies
          Installing node modules (package.json)
   express@3.4.8 node_modules/express
   ├── methods@0.1.0
   ├── debug@0.8.1
   ├── merge-descriptors@0.0.1
   ├── cookie-signature@1.0.1
   ├── fresh@0.2.0
   ├── range-parser@0.0.4
   ├── buffer-crc32@0.2.1
   ├── cookie@0.1.0
   ├── mkdirp@0.3.5
   ├── commander@1.3.2 (keypress@0.1.0)
   ├── send@0.1.4 (mime@1.2.11)
```

```
   └── connect@2.12.0 (uid2@0.0.3, qs@0.6.6, pause@0.0.1, raw-body@1.1.2, bytes@0.2.1, batch@0.5.0,
negotiator@0.3.0, multiparty@2.2.0)
   -----> Caching build
          Clearing previous node cache
          Saving 3 cacheDirectories (default):
          - .npm
          - .cache/yarn (nothing to cache)
          - bower_components (nothing to cache)
          **WARNING** This app may not specify any way to start a node process
          See: https://docs.cloudfoundry.org/buildpacks/node/node-tips.html#start
   Exit status 0
   Uploading droplet, build artifacts cache...
   Uploading droplet...
   Uploading build artifacts cache...
   Uploaded build artifacts cache (3.8M)
   Uploaded droplet (13.6M)
   Uploading complete
   Cell 427f4537-d113-4050-8f1f-eb1875759ae8 stopping instance b68559a4-73cb-4f68-b981-3b4c9441f7ae
   Cell 427f4537-d113-4050-8f1f-eb1875759ae8 destroying container for instance b68559a4-73cb-4f68-b981-3b4c9441f7ae

Waiting for app to start...
   Cell 427f4537-d113-4050-8f1f-eb1875759ae8 successfully destroyed container for instance b68559a4-73cb-4f68-b981-
3b4c9441f7ae

name:              cftest
requested state:   started
instances:         1/1
usage:             512M x 1 instances
routes:            cftest-1234.cfapps.io
last uploaded:     Thu 29 Mar 20:46:10 PDT 2018
stack:             cflinuxfs2
buildpack:         nodejs
start command:     node app.js

     state     since                  cpu    memory          disk       details
#0   running   2018-03-30T03:46:37Z   0.0%   41.4K of 512M   8K of 1G

user@ubuntu:~/cfar$
```

It worked!

Note that PWS has made our app available at: cftest-1234.cfapps.io. The CF router will automatically direct traffic targeting this DNS name to our application on its listening port.

Try to reach it:

```
user@ubuntu:~/cfar$ curl http://cftest-1234.cfapps.io/status

Server A

user@ubuntu:~/cfar$
```

Your app is running in the cloud!

7. Exploring further

List all of the application running in your space:

```
user@ubuntu:~/cfar$ cf apps

Getting apps in org MontyAnaconda / space development as randy.abernethy@gmail.com...
OK

name     requested state   instances   memory   disk   urls
cftest   started           1/1         512M     1G     cftest-1234.cfapps.io

user@ubuntu:~/cfar$
```

List your applications logs:

```
user@ubuntu:~/cfar$ cf logs cftest --recent
Retrieving logs for app cftest in org MontyAnaconda / space development as randy.abernethy@gmail.com...

...
```

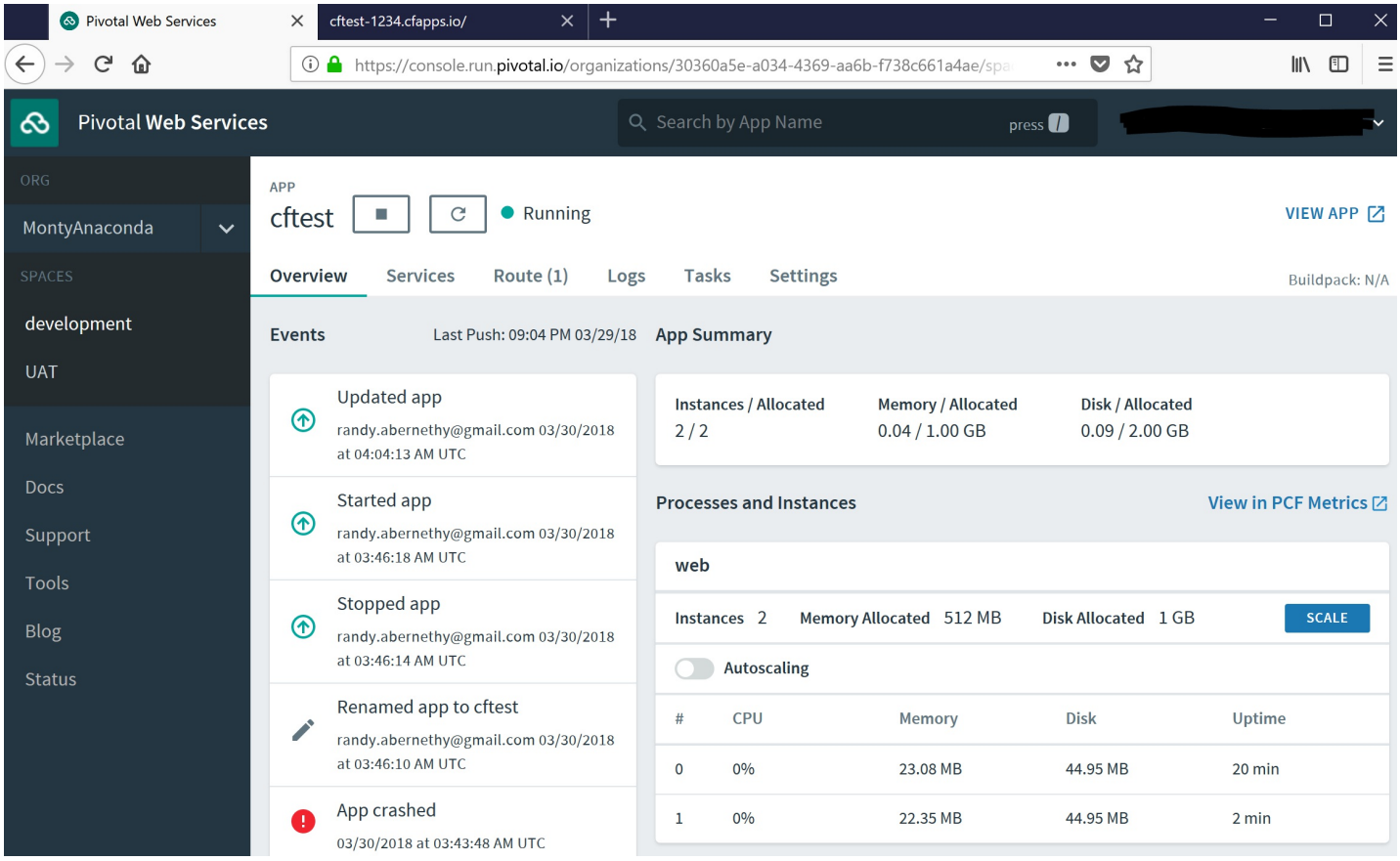
```
2018-03-29T21:01:03.94-0700 [RTR/1] OUT cftest-1234.cfapps.io - [2018-03-30T04:01:03.938+0000] "GET /status
HTTP/1.1" 200 0 9 "-" "curl/7.47.0" "10.10.2.198:44644" "10.10.149.92:61066" x_forwarded_for:"38.128.194.54,
10.10.2.198" x_forwarded_proto:"http" vcap_request_id:"1b500eef-6947-4be3-42e0-d48c3e2f9d31"
response_time:0.009077704 app_id:"982a502c-9684-4b21-a1a9-92b9456df30d" app_index:"0"
x_b3_traceid:"62d76e2ec23d5a8f" x_b3_spanid:"62d76e2ec23d5a8f" x_b3_parentspanid:"-"
2018-03-29T21:01:03.94-0700 [RTR/1] OUT
```

This displays the recent log output from your application. The last thin in the output should be the log for the curl test we ran, "GET /status HTTP/1.1".

Now try scaling you app up to two instance:

```
user@ubuntu:~/cfar$ cf scale cftest -i 2

Scaling app cftest in org MontyAnaconda / space development as randy.abernethy@gmail.com...
OK

user@ubuntu:~/cfar$ cf apps

Getting apps in org MontyAnaconda / space development as randy.abernethy@gmail.com...
OK

name     requested state   instances   memory   disk   urls
cftest   started           2/2         512M     1G     cftest-1234.cfapps.io

user@ubuntu:~/cfar$
```

Switch to the Web Console and locate you app. Click your apps name in the space app table to see detailed output:



Now stop your app:

```
user@ubuntu:~/cfar$ cf stop cftest

Stopping app cftest in org MontyAnaconda / space development as randy.abernethy@gmail.com...
OK

user@ubuntu:~/cfar$
```

Try to curl it:

```
user@ubuntu:~/cfar$ curl http://cftest-1234.cfapps.io/status

404 Not Found: Requested route ('cftest-1234.cfapps.io') does not exist.

user@ubuntu:~/cfar$
```

Nobody home. Now restart the app and then try to curl it again:

```
user@ubuntu:~/cfar$ cf start cftest

Starting app cftest in org MontyAnaconda / space development as randy.abernethy@gmail.com...

Waiting for app to start...

name:              cftest
requested state:   started
instances:         2/2
usage:             512M x 2 instances
routes:            cftest-1234.cfapps.io
last uploaded:     Thu 29 Mar 20:46:10 PDT 2018
stack:             cflinuxfs2
buildpack:         nodejs
start command:     node app.js

     state     since                  cpu    memory          disk        details
#0   running   2018-03-30T04:09:32Z   0.1%   23.1M of 512M   45M of 1G
#1   running   2018-03-30T04:09:33Z   0.2%   22.3M of 512M   45M of 1G

user@ubuntu:~/cfar$ curl http://cftest-1234.cfapps.io/status

Server A

user@ubuntu:~/cfar$
```

To complete the lab stop you app.

```
user@ubuntu:~/cfar$ cf stop cftest

Stopping app cftest in org MontyAnaconda / space development as randy.abernethy@gmail.com...
OK

user@ubuntu:~/cfar$
```

Congratulations, you have completed the Lab.