m. Set up or used configuration management

BE typically leverages tools such as Chef or Puppet or those available from our customers for "configuration management" of Enterprise applications and the environments they run in. We also enforce strict processes for controlling the configuration of our environments in order to avoid the problems that come with mismatched or unknown configurations. For the purposes of this prototype BE felt it important to ensure a consistent platform on which to deploy its prototype application and using just enough technology to ensure the right configuration control. BE opted to use Amazon Web Service's Elastic Beanstalk, AWS Config and Docker to provide just the right amount of configuration management for this prototype.

Using Elastic Beanstalk allowed BE to select the EC2 instance running 64bit Amazon Linux 2015.03 v1.4.3 running Docker 1.6.2 and consistently deploy it every time in a Docker container. BE would reevaluate the configuration management needs for this application should it be taken beyond a prototype phase.

AWS Config provided BE with an inventory of our AWS instances and logs of their configurations. It pushes notifications to administrators whenever those instances are modified. AWS Config allows BE an inventory of AWS resources to be exported with all of the details regarding their configurations. This allows BE to resolve how an instance was configured at any given point in time during its lifespan. Such capabilities enable BE to track resources utilized, analyze security, troubleshoot issues, and audit the compliance of their instances.

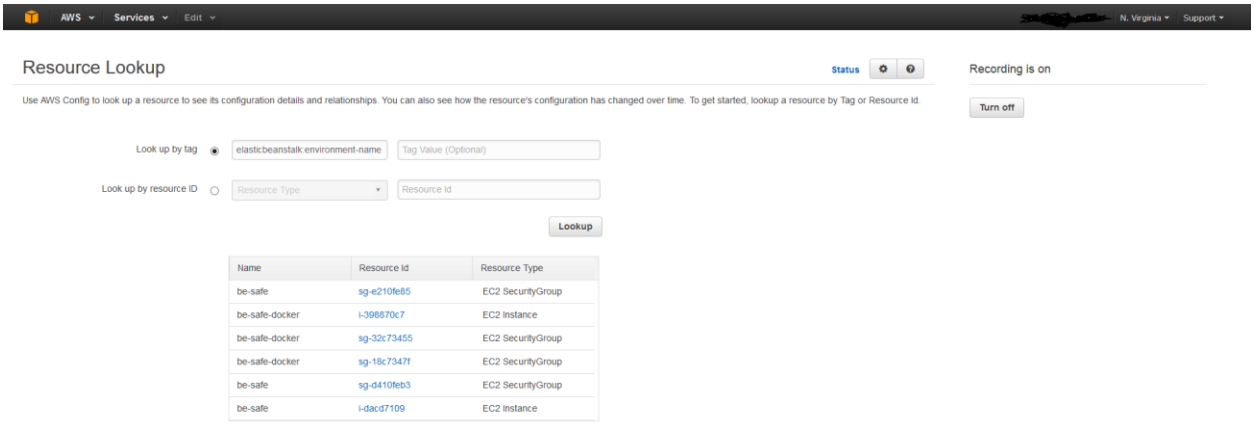

Figure 1. AWS Config view

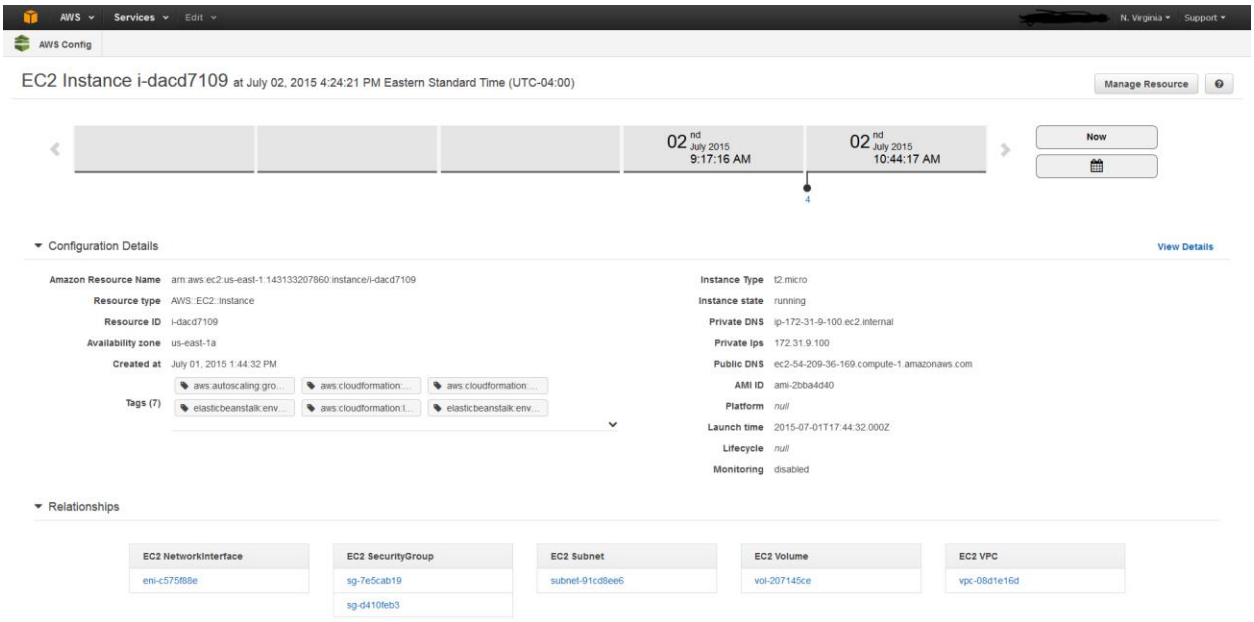

Figure 2. AWS Config resource view

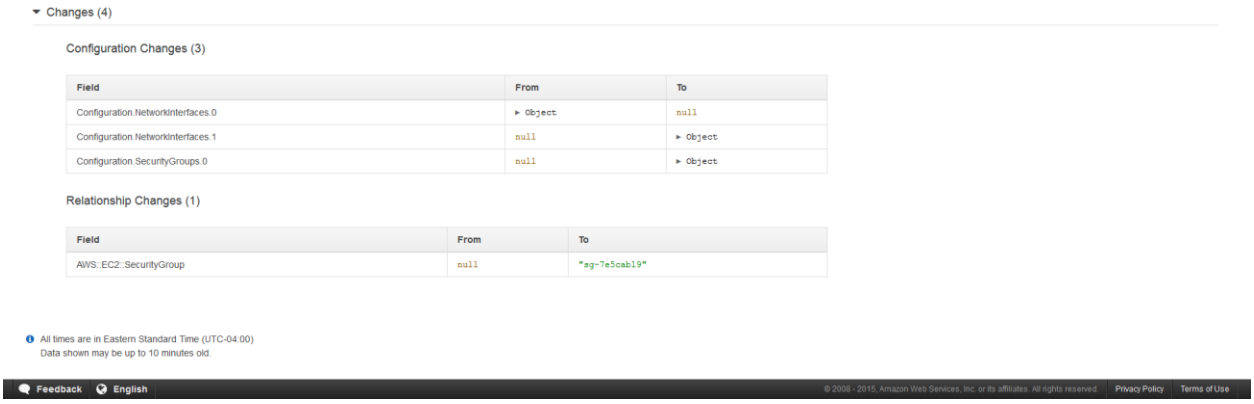

Figure 3. AWS Config resource view

We can also view additional technical details of our instance:

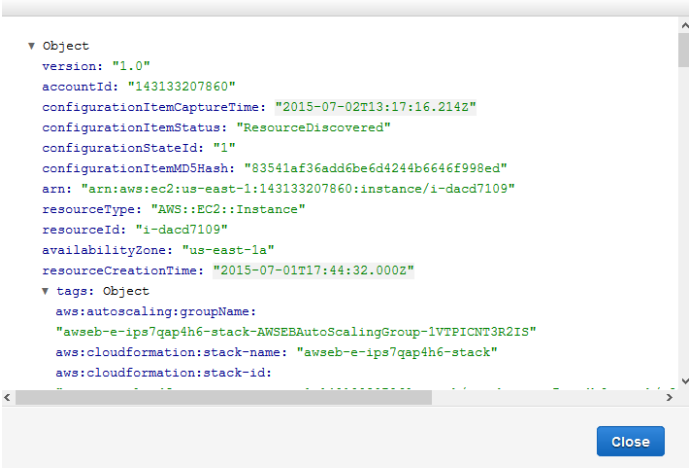

Figure 4. Viewing Additional Technical Details of the Instance

In addition to Elastic Beanstalk we leveraged the DevOps principle of Infrastructure as code where we scripted the build of the Docker Container during the build and deployment process. Building and deploying the container with code allows us tighter configuration control of our environment for example it allows us to add dependencies in the Docker container simply by modifying the build script.

Figure 5 shows the BE Safe Elastic Beanstalk console running a Docker container.

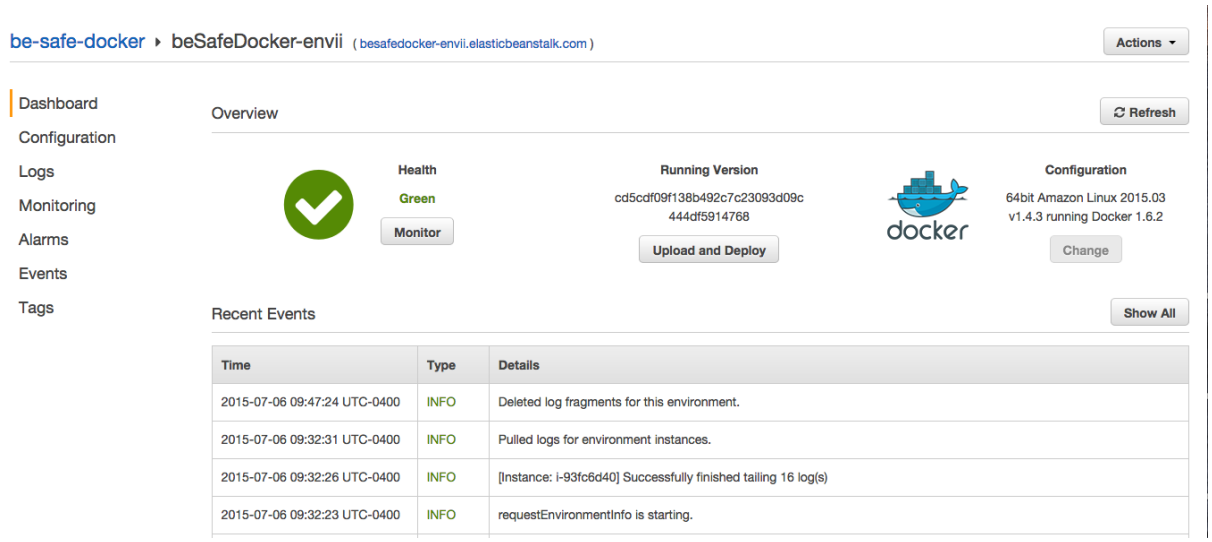

Figure 5. Elastic Beanstalk console running a Docker container

Figure 6 shows the Dockerfile file used to build and configure the Docker container.

```
#Set Container OS
FROM ubuntu:14.04

#Expose a port
EXPOSE 3000

#Prepare the environment
RUN sudo apt-get -y install curl
RUN curl --silent --location https://deb.nodesource.com/setup_0.12 | sudo bash -
RUN sudo apt-get -y install nodejs git python2.7 make g++

#Set up git
RUN git config --global user.email "be-safe@buchanan-edwards.com"
RUN git config --global user.name "BE Safe User"

#Clone the most current branch of the github repo
RUN git clone -b dev https://github.com/brandtheisey/BE-Safe.git besafe/

#Set up the application's node dependencies
RUN cd /besafe && npm install -g gulp
RUN cd /besafe && npm install
RUN cd /besafe && gulp app
```

Figure 6. Dockerfile file

In this setup BE uploaded our code and allowed Elastic Beanstalk to automatically handle the deployment of servers, software, capacity, load balancing, auto-scaling and application health monitoring. For a short prototype development project utilizing GitHub and continuous integration, we had no further requirements dictating the use of a tool such as Chef or Puppet.