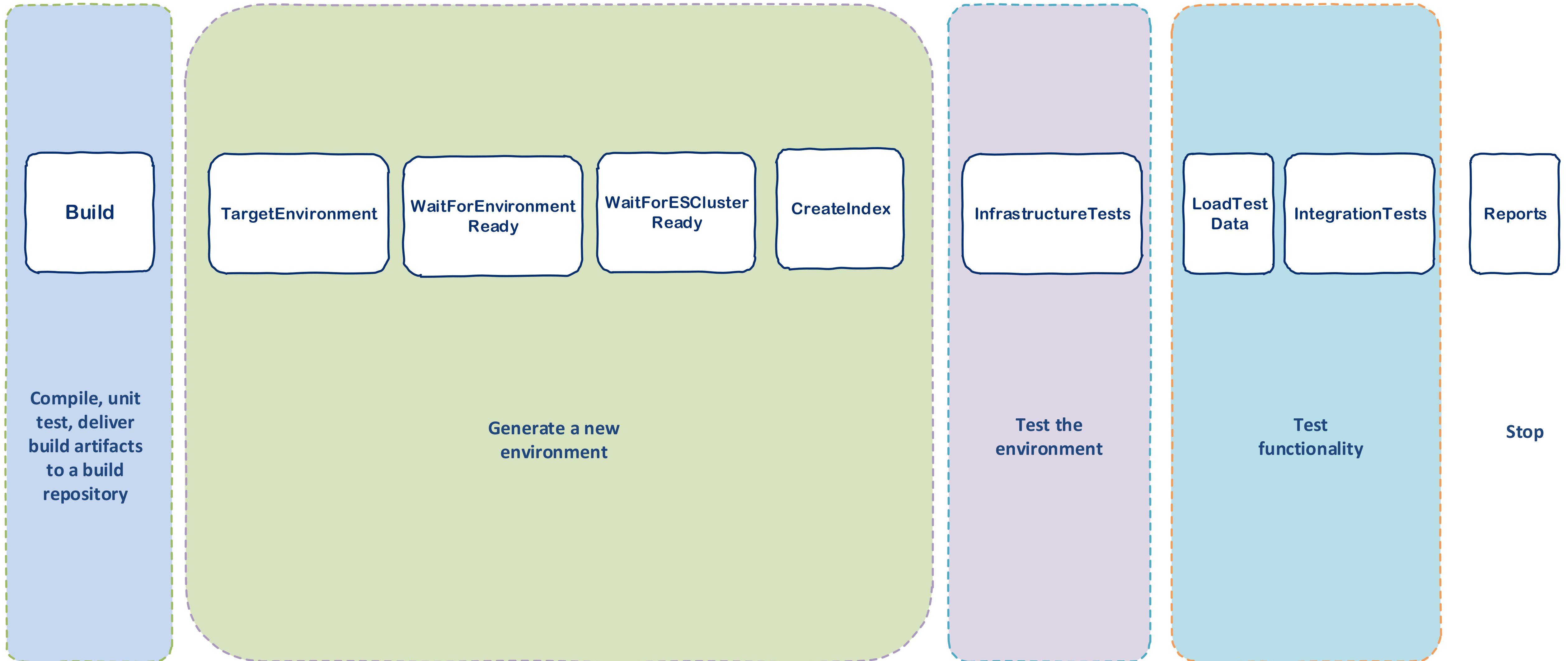
# Proteus CD Steps

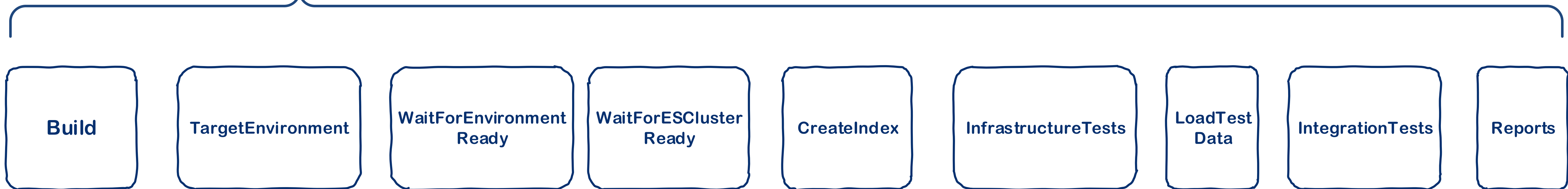
These are the actual pipeline steps you'll see for Proteus defined in Jenkins. The steps are mapped to high level CD areas.



# Tech Snapshot

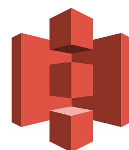
A brief look at the technologies used to realize the Proteus CD flow.

Jenkins is used to orchestrate the entire flow. Jenkins manages the critical version number that is applied to all artifacts. It also handles passing variables from one step to another.



Source control

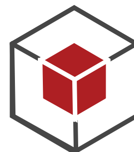
msbuild



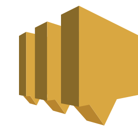
Amazon S3

Code repo

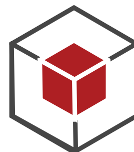
AWS CloudFormation



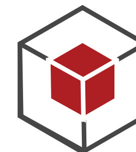
Ruby



Amazon SNS



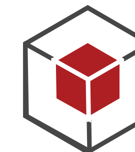
Ruby



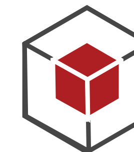
Ruby

Ruby

Infrastructure tests written using serverspec. Tests are launched from a Ruby script.



Ruby



Ruby

Integration tests written using Cucumber. Tests are launched from a Ruby script, which passes dynamic variables.

# Build Details

~ 45 seconds

All code lives in a private repository in **GitHub**. Code includes app code, infrastructure code, and test code.

GitHub was selected because we did not have a VPN connection to CTC at the time. We also read guidance from other companies that TFS is not as Jenkins friendly as GitHub.

**Jenkins** has a GitHub hook. Any change in code triggers a build.

Our builder is msbuild, no different than TFS.

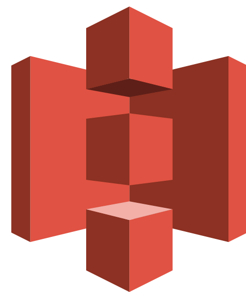
Jenkins names the build and assigns the critical **system wide version** for all artifacts.

${ENV,var="GIT_BRANCH"}-1.0.${BUILD_NUMBER}

Build artifacts are **zipped** up and stored in S3.

ss_v1.0.${BUILD_NUMBER}.zip

Jenkins notifies via email and Jabber.



Amazon S3

artifacts.proteusa.com/SearchServices

# TargetEnvironment

~ 3 – 7 mins

Jenkin creates an AWS Cloud Formation stack. The CF templates live in GitHub.

Jenkins generates the name of the new stack by passing in the build number variable.

**Apx-Development-v-${SRS_BUILD_NUMBER}**

Jenkins passes the following variables to CF:

PublicBucket=${PublicBucket}
KeyName=${KeyName}
HostedZone=${HostedZone}
EnvironmentVersion=${SRS_BUILD_NUMBER}
IndexName=${IndexName}

AWS CloudFormation

Base infrastructure configuration.

Installs and runs Puppet.

```
service puppet start
```

Runs our Ruby script for EnvReady

```
nohup ruby ~/check_puppet.rb
```

***Pattern***: What's in CF versus Puppet?

CloudFormation manages the architecture and layout of our AWS resources.

Puppet manages the configurations, services and applications within those machines.

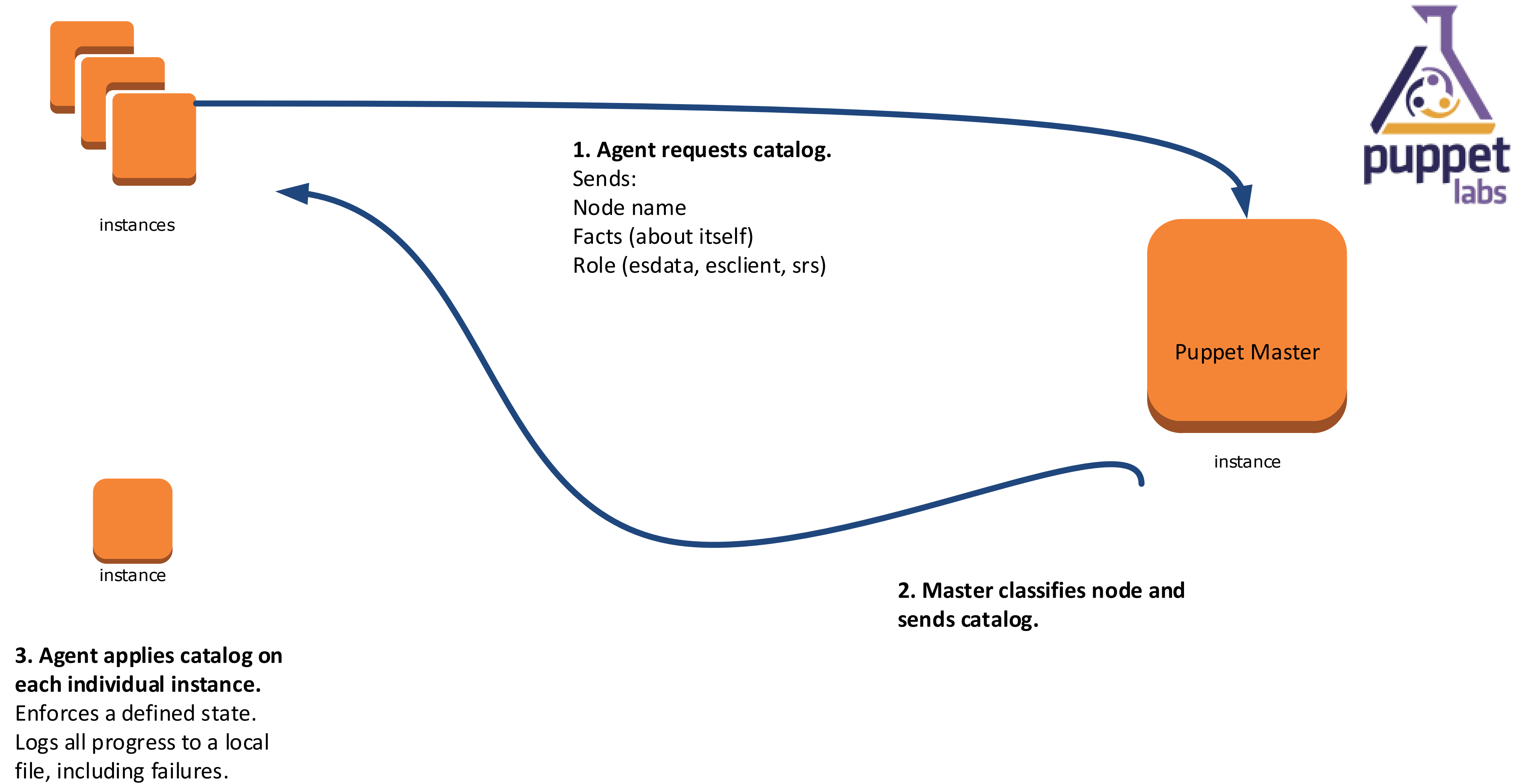
Consistent with Amazon's thinking:
https://s3.amazonaws.com/cloudformation-examples/IntegratingAWSCloudFormationWithPuppet.pdf

**Base Images TBD**

**Start state for Puppet**
CloudFormation spins up the infrastructure, installs and runs the puppet agent.

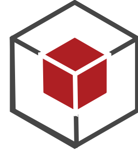
# Puppet

This diagram describes what is happening during the next CD step (WaitForEnvReady)



# WaitForEnvironmentReady

~ 10 – 20 mins

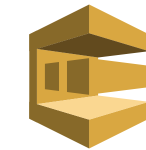


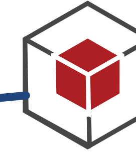

Jenkins runs a Ruby script that polls an Amazon queue for messages posted from the EC2 instances.

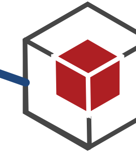
Jenkins passes the stack name and minimum number of instances to this script.

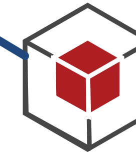
If the script finds the minimum number of instances post to the queue and there are no errors from any machine posted, this pipeline step passes.

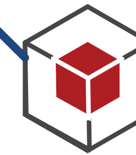
This build step will time out in 20 minutes.









. . .

This Ruby script listens for a file that indicates Puppet is done with its work.

/var/lib/puppet/state/last_run_summary.yaml

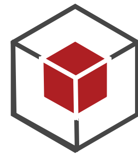
Reads the local Puppet agent error log and posts to the Amazon topic including any errors.

***Todo***: Use MCollective and back off the custom code.

Probably too much custom code for the long run. Other people are using Puppet MCollective to evaluate environment 'doneness'. MCollective appears to work in a similar way to our solution.

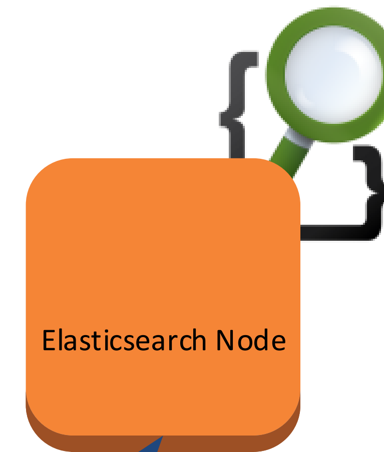
# WaitForESClusterReady

~ 5 seconds




Jenkins calls the es_cluster_state Ruby script passing in parameters for the stack name, URL of the newly formed ES cluster, and the minimum number of ES nodes that should be available. This script polls the ES cluster state API for a configurable amount of time. If the cluster is green AND the min number of nodes are available, this step passes.

```
GET 'http://host:9200/_cluster/state'
```

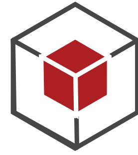




```
{
    cluster_name: "Apx-Development-v-382",
    status: "green",
    timed_out: false,
    number_of_nodes: 6,
    number_of_data_nodes: 4,
    active_primary_shards: 2,
    active_shards: 8,
    relocating_shards: 0,
    initializing_shards: 0,
    unassigned_shards: 0
}
```

# CreateIndex

~ 9 seconds



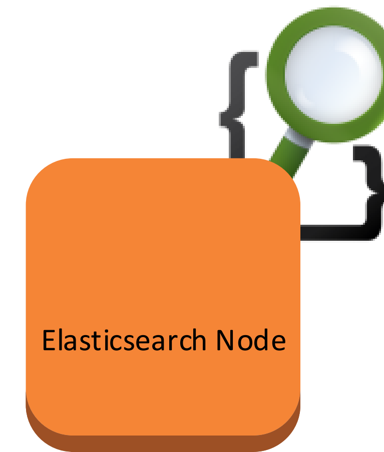
Ruby

Jenkins calls a ruby script named es_create_index and passes in the following variables:

- stack name
- index name
- mapping file location

These Ruby scripts use an Elasticsearch Ruby gem to simplify interaction with the search engine.

https://github.com/elasticsearch/elasticsearch-ruby



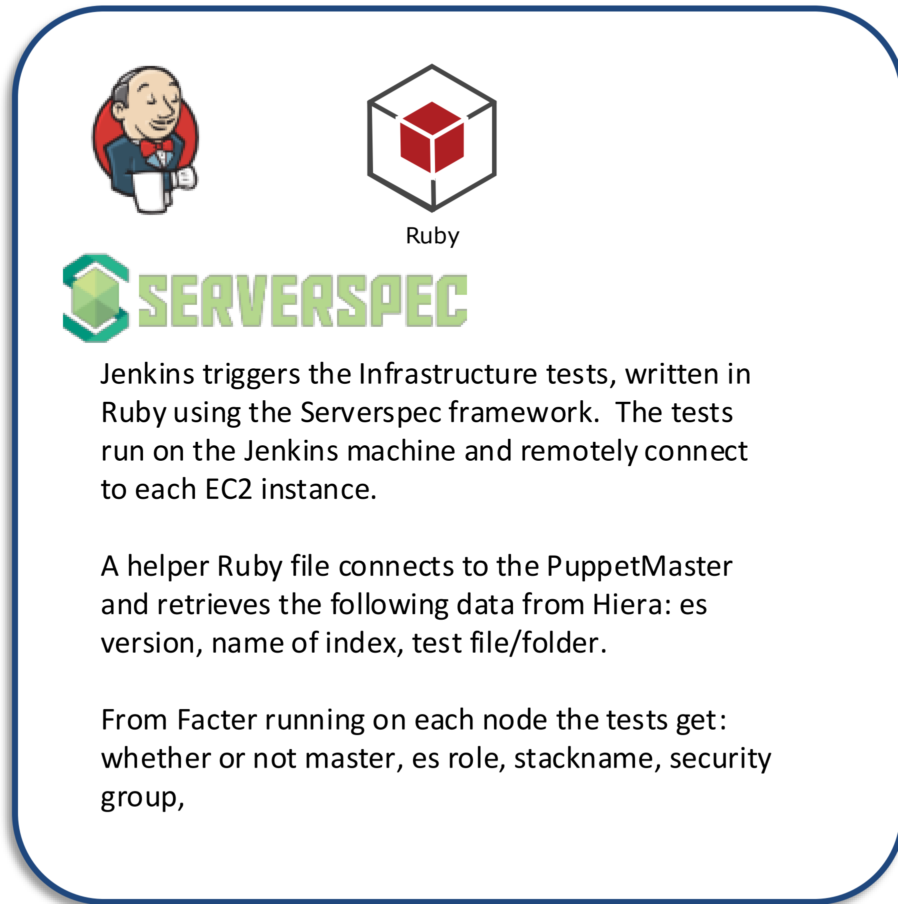
Elasticsearch Node

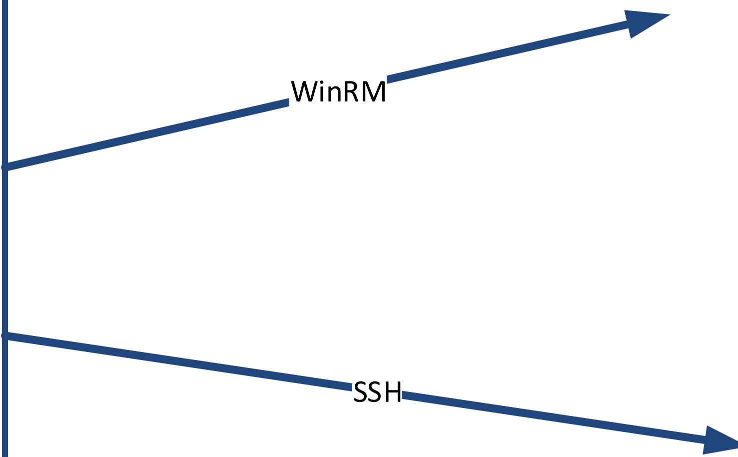
instance

# InfrastructureTests

~ 2 minutes 45 secs

```ruby
describe package('java-1.7.0-openjdk.x86_64') do
  it { should be_installed }
end
```



instances
Windows Search Service

instances
Linux ES Client

instances
Linux ES Data

Passing variables: Hiera versus Facter

Hiera data will not spin a new build. Facter will. Put everything we can in Hiera unless it should spin a new build.

Facter is stuff that is unique to a machine or release.

# LoadTestData

~ 4 minutes

Jenkins executes a Windows executable that we wrote to load test data into the ES cluster. This executable needs to know:

* The name of the current infrastructure stack
* The format of content (appl)
* The location where we will look up items
* The location where our test content is defined.

1

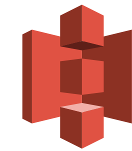
The list of itemIds we use to load the index is defined by QA. It lives in GitHub and is maintained by the team. This list represents the **controlled** set of content required for our automated tests.

509 lines (508 sloc) | 17.815 kb

```
1  3a5d97dc98bb48b7aed7463f20e510e7.0
2  62ff17659fed4689a9dc7edc73a4b6e6.0
3  cdeee2083acc41ffb39f79e45548c797.0
4  bbcaed565cb54faaa02a38597c38ca22.0
5  246d6e50595647ec80acbebd1289e197.2
```

2

ECR



Amazon S3

For each item.RSN, the EXE pulls the content from the ECR.

3

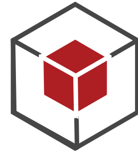
Each appl document is mapped to appl-json using the same mapping code as ASPEN. The EXE then pushes the content to the ES cluster.

Elasticsearch Node

instance

# IntegrationTests

~ 2.5 minutes



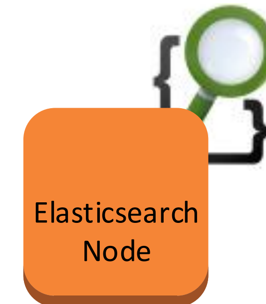
Jenkins calls the QA or Integration tests. The tests are written in Cucumber, a behavioral driven development (BDD) framework built on top of Ruby.

cctest.rb is run locally on the Jenkins server. Several variables are passed to this script including the aws stack name. The script uses this name and the aws-sdk to pull the new URL for the search service. This enables the Cucumber tests to target the newly spun search environment.

```
@apx @itemid
Scenario: Execute an itemid apx search and check the itemid is returned
  When an apx search for itemid is performed
  Then the result is the requested itemid
```

```
raise "ItemId in response ('#{itemid}') does not match expected ('#{@itemid}')" if @itemid != itemid
```

**1: Cucumber tests run**




**2: Results logged locally**

# Success