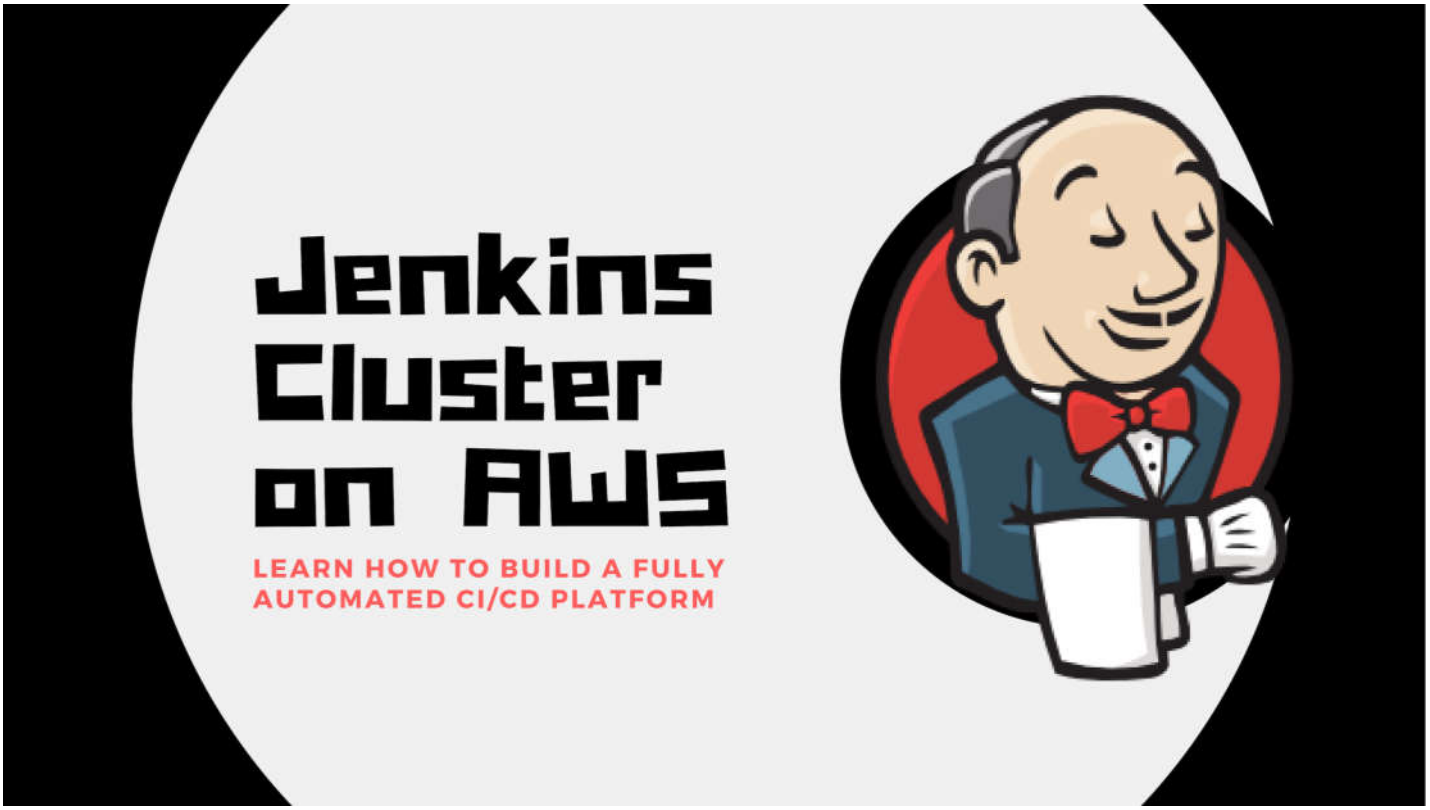
Deploy a Jenkins Cluster on AWS

The automated platform has a Jenkins cluster with a dedicated Jenkins master and workers inside an autoscaling group

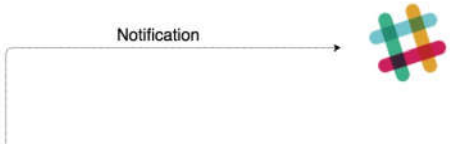
Mohamed Labouardy
Nov 4, 2018 · 6 min read

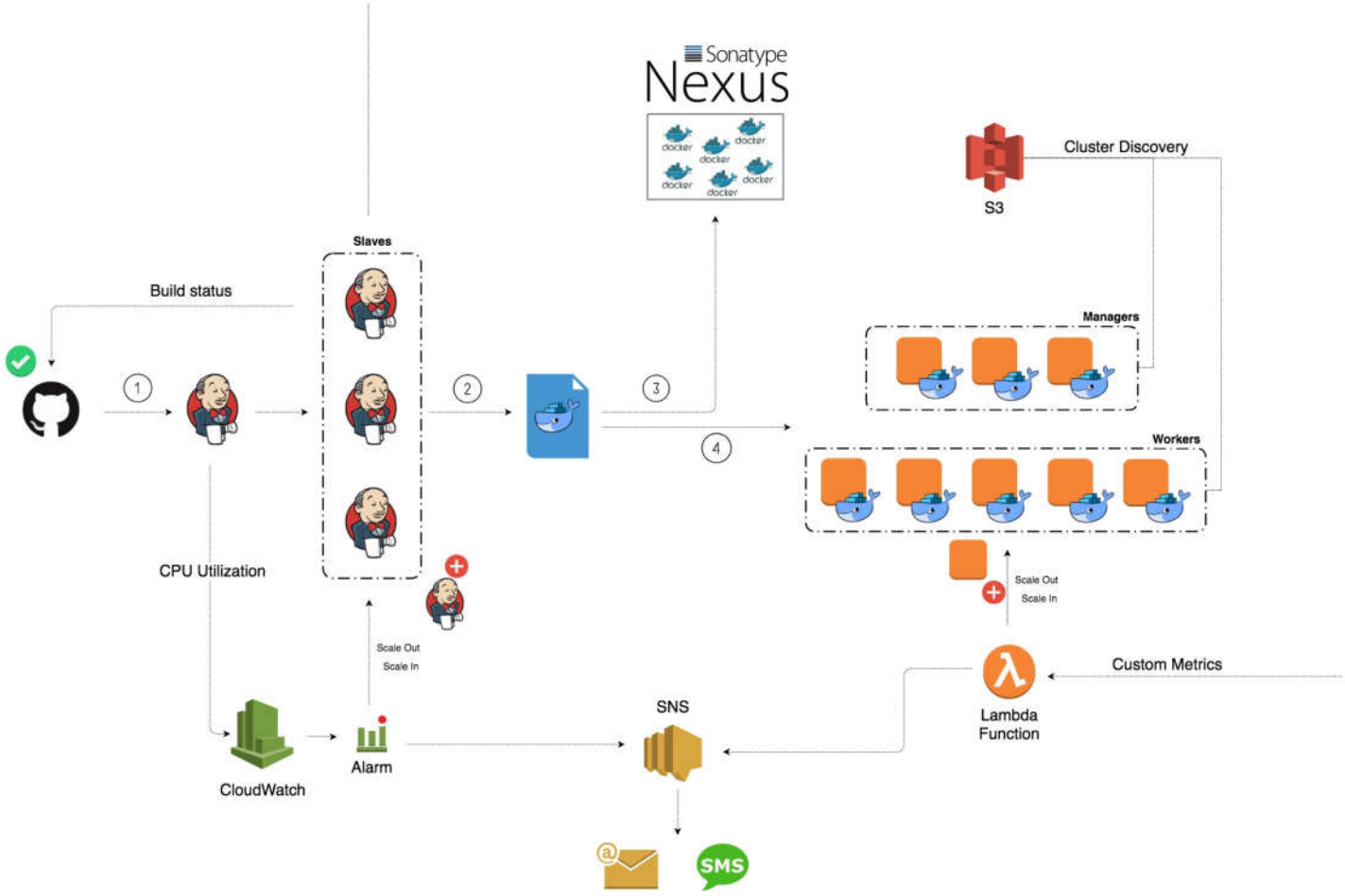


A few months ago, I gave a talk at Nexus User Conference 2018 on how to build a fully automated CI/CD platform on AWS using Terraform, Packer & Ansible.

The session illustrated how concepts like *infrastructure as code*, *immutable infrastructure*, *serverless*, *cluster discovery*, etc can be used to build a highly available and cost-effective pipeline.

The platform I built is represented in the following diagram:





The platform has a Jenkins cluster with a dedicated Jenkins master and workers inside an autoscaling group. Each push event to the code repository will trigger the Jenkins master which will schedule a new build on one of the available slave nodes.

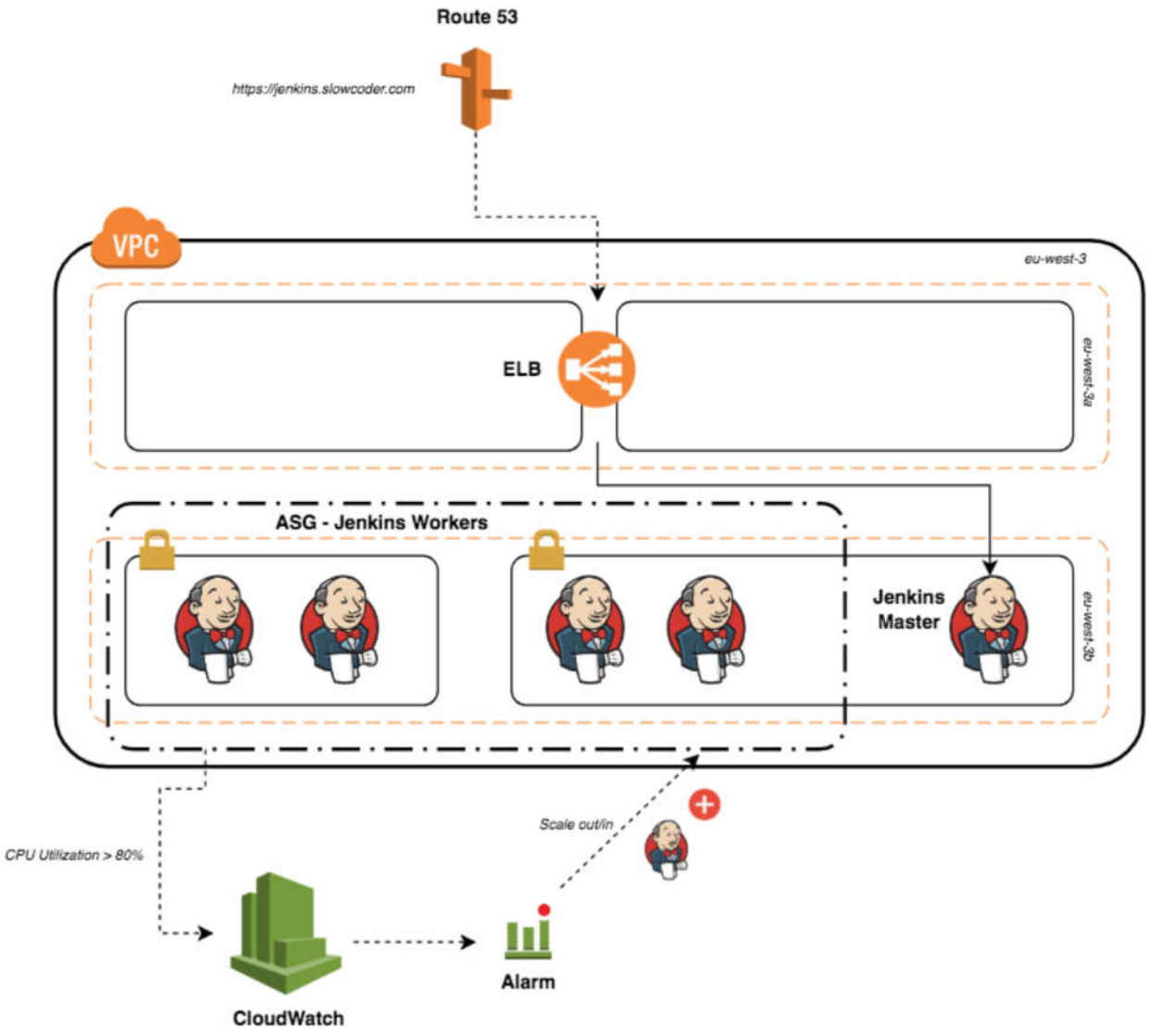
The slave nodes will be responsible of running the unit and pre-integration tests, building the Docker image, storing the image to a private registry and deploying a container based on that image to Docker Swarm cluster.

If you missed my talk, you can watched it again on YouTube

On this post, I will walk through how to deploy the Jenkins cluster on AWS using the latest automation tools.

The cluster will be deployed into a VPC with 2 public and 2 private subnets across 2 availability zones. The stack will consists of an autoscaling group of Jenkins workers in a private subnets and a private instance for the Jenkins master sitting behind an elastic Load balancer.

To add or remove Jenkins workers on-demand, the CPU utilization of the ASG will be used to trigger a scale out (*CPU > 80%*) or scale in (*CPU < 20%*) event.



To get started, we will create 2 AMIs (Amazon Machine Image) for our instances. To do so, we will use *Packer*, which allows you to bake your own image.

The first AMI will be used to create the Jenkins master instance. The AMI uses the *Amazon Linux Image* as a base image and for provisioning part it uses a simple shell script:

The shell script will be used to install the necessary dependencies, packages and security patches:

It will install the latest stable version of Jenkins and configure its settings:

- Create a Jenkins admin user.

- Create a SSH, GitHub and Docker registry credentials.
- Install all needed plugins (Pipeline, Git plugin, Multi-branch Project, etc).
- Disable remote CLI, JNLP and unnecessary protocols.
- Enable CSRF (Cross Site Request Forgery) protection.
- Install Telegraf agent for collecting resource and Docker metrics.

The second AMI will be used to create the Jenkins workers, similarly to the first AMI, it will be using the Amazon Linux Image as a base image and a script to provision the instance:

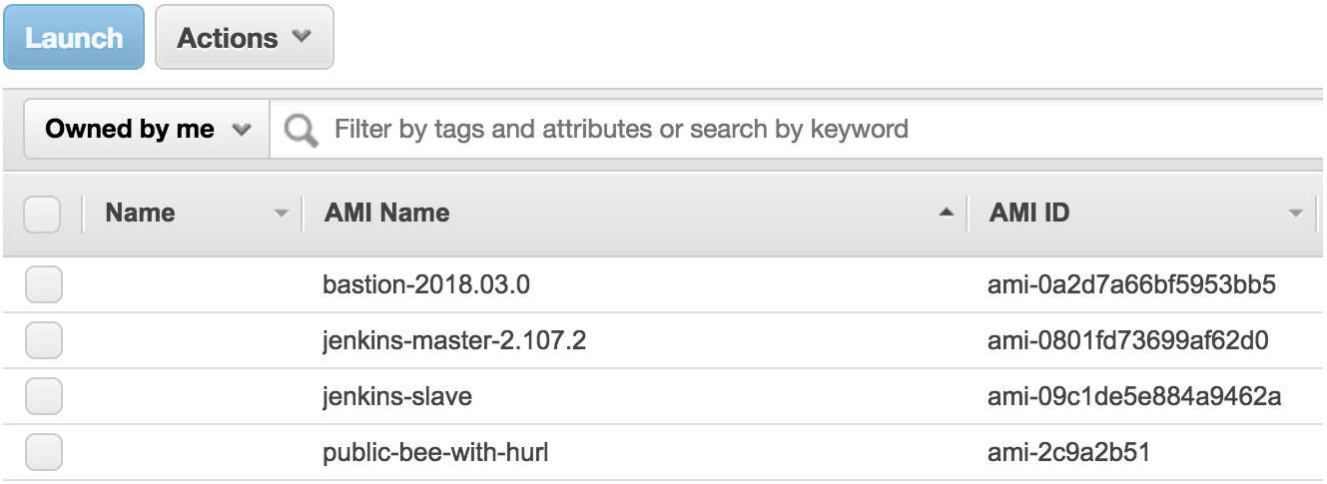
A Jenkins worker requires the Java JDK environment and Git to be installed. In addition, the Docker community edition (building Docker images) and a data collector (monitoring) will be installed.

Now our Packer template files are defined, issue the following commands to start baking the AMIs:

Packer will launch a temporary EC2 instance from the base image specified in the template file and provision the instance with the given shell script. Finally, it will create an image from the instance. The following is an example of the output:

```
amazon-ebs output will be in this color.

==> amazon-ebs: Prevalidating AMI Name: jenkins-master-2.107.2
    amazon-ebs: Found Image ID: ami-0ebc281c20e89ba4b
==> amazon-ebs: Creating temporary keypair: packer_5bddcc14-a599-c961-d7a6-c18a1954881c
==> amazon-ebs: Creating temporary security group for this instance: packer_5bddcc15-3af0-5af6-fc7d-ba5d413ccc90
==> amazon-ebs: Authorizing access to port 22 from 0.0.0.0/0 in the temporary security group...
==> amazon-ebs: Launching a source AWS instance...
==> amazon-ebs: Adding tags to source instance
    amazon-ebs: Adding tag: "Name": "packer-builder-docker"
    amazon-ebs: Instance ID: i-042b4ef9942ea5539
==> amazon-ebs: Waiting for instance (i-042b4ef9942ea5539) to become ready...
```

Sign in to AWS Management Console, navigate to "**EC2 Dashboard**" and click on "**AMI**", 2 new AMIs should be created as below:



Now our AMIs are ready to use, let's deploy our Jenkins cluster to AWS. To achieve that, we will use an infrastructure as code tool called *Terraform*, it allows you to describe your entire infrastructure in templates files.

I have divided each component of my infrastructure to a template file. The following template file is responsible of creating an EC2 instance from the Jenkins master's AMI built earlier:

Another template file used as a reference to each *AMI* built with *Packer*:

The Jenkins workers (aka *slaves*) will be inside an autoscaling group of a minimum of 3 instances. The instances will be created from a launch configuration based on the Jenkins slave's AMI:

To leverage the power of automation, we will make the worker instance join the cluster automatically (*cluster discovery*) using Jenkins RESTful API:

At boot time, the *user-data* script above will be invoked and the instance private IP address will be retrieved from the instance *meta-data* and a groovy script will be executed to make the node join the cluster:

Moreover, to be able to *scale out* and *scale in* instances on demand, I have defined 2 *CloudWatch* metric alarms based on the CPU utilisation of the autoscaling group:

Finally, an *Elastic Load Balancer* will be created in front of the Jenkins master's instance and a new DNS record pointing to the *ELB* domain will be added to *Route 53*:

Once the stack is defined, provision the infrastructure with *terraform apply* command:

The command takes an additional parameter, a variables file with the AWS credentials and VPC settings (You can create a new VPC with Terraform from here):

Terraform will display an *execution plan* (list of resources that will be created in advance), type *yes* to confirm and the stack will be created in few seconds:

```
aws_instance.jenkins_master: Creating...
  ami:                                      "" => "ami-0801fd73699af62d0"
  arn:                                      "" => "<computed>"
  associate_public_ip_address:              "" => "<computed>"
  availability_zone:                        "" => "<computed>"
  cpu_core_count:                           "" => "<computed>"
  cpu_threads_per_core:                     "" => "<computed>"
  ebs_block_device.#:                       "" => "<computed>"
  ephemeral_block_device.#:                 "" => "<computed>"
  get_password_data:                        "" => "false"
  instance_state:                           "" => "<computed>"
  instance_type:                            "" => "t2.large"
  ipv6_address_count:                       "" => "<computed>"
  ipv6_addresses.#:                         "" => "<computed>"
  key_name:                                 "" => "demo"
  network_interface.#:                      "" => "<computed>"
  network_interface_id:                     "" => "<computed>"
  password_data:                            "" => "<computed>"
  placement_group:                          "" => "<computed>"
  primary_network_interface_id:             "" => "<computed>"
  private_dns:                              "" => "<computed>"
  private_ip:                               "" => "<computed>"
  public_dns:                               "" => "<computed>"
  public_ip:                                "" => "<computed>"
  root_block_device.#:                      "" => "1"
  root_block_device.0.delete_on_termination: "" => "false"
  root_block_device.0.volume_id:            "" => "<computed>"
  root_block_device.0.volume_size:          "" => "30"
  root_block_device.0.volume_type:          "" => "gp2"
  security_groups.#:                        "" => "<computed>"
  source_dest_check:                        "" => "true"
  subnet_id:                                "" => "subnet-0d8840d5f3be0cd46"
  tags.%:                                   "" => "3"
  tags.Author:                              "" => "nexus-user-conference"
  tags.Name:                                "" => "jenkins_master"
  tags.Tool:                                "" => "Terraform"
  tenancy:                                  "" => "<computed>"
  volume_tags.%:                            "" => "<computed>"
  vpc_security_group_ids.#:                 "" => "1"
  vpc_security_group_ids.3677025563:        "" => "sg-07cc5962d08620d36"
```

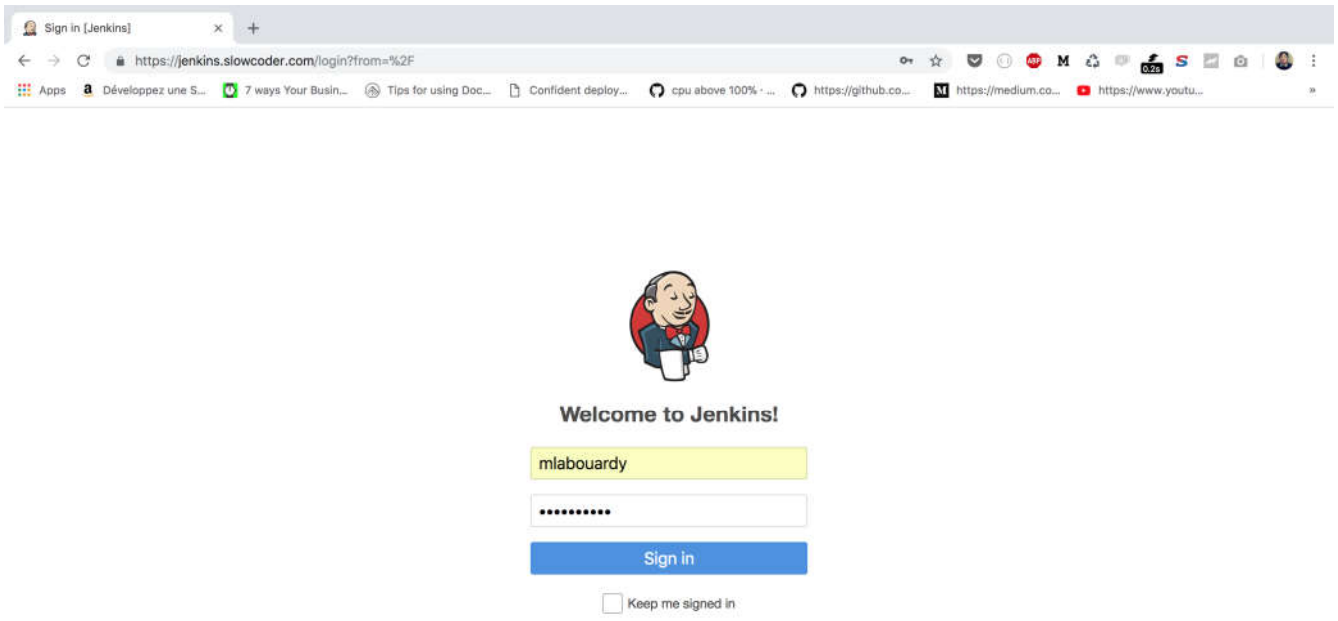
Jump back to EC2 dashboards, a list of EC2 instances will created:

jenkins_slave	i-0539f57faa7ba68a5	t2.micro	eu-west-3a	running	Initializing	None	
jenkins_slave	i-06949e5c6758711c5	t2.micro	eu-west-3b	running	Initializing	None	
nexus	i-0765c120ea27e0a25	t2.xlarge	eu-west-3a	running	Initializing	None	

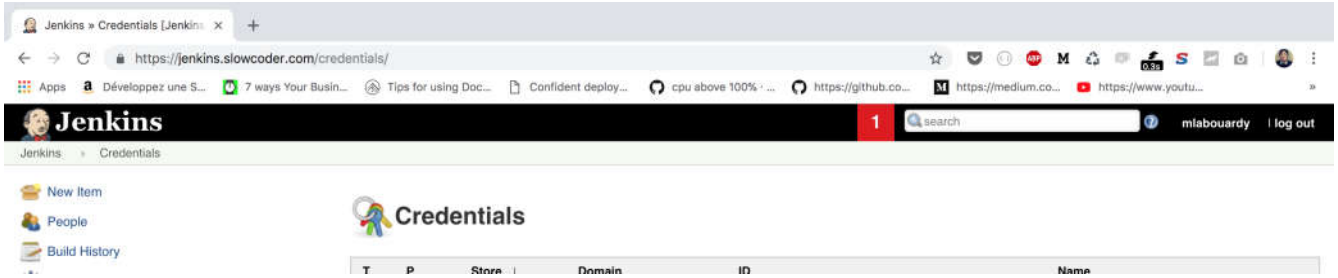
In the terminal session, under the *Outputs* section, the Jenkins URL will be displayed:

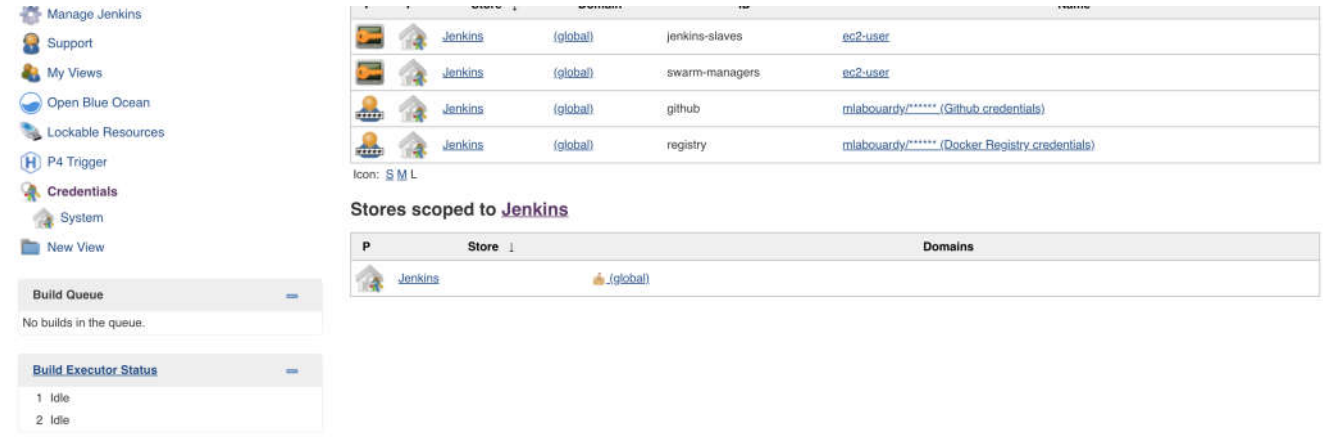
```
Apply complete! Resources: 14 added, 0 changed, 0 destroyed.

Outputs:

Jenkins DNS = https://jenkins.slowcoder.com
Jenkins SG ID = sg-07cc5962d08620d36
Nexus DNS = https://nexus.slowcoder.com
Registry DNS = https://registry.slowcoder.com
```

Point your favorite browser to the URL displayed, the Jenkins login screen will be displayed. Sign in using the credentials provided while baking the Jenkins master's AMI:

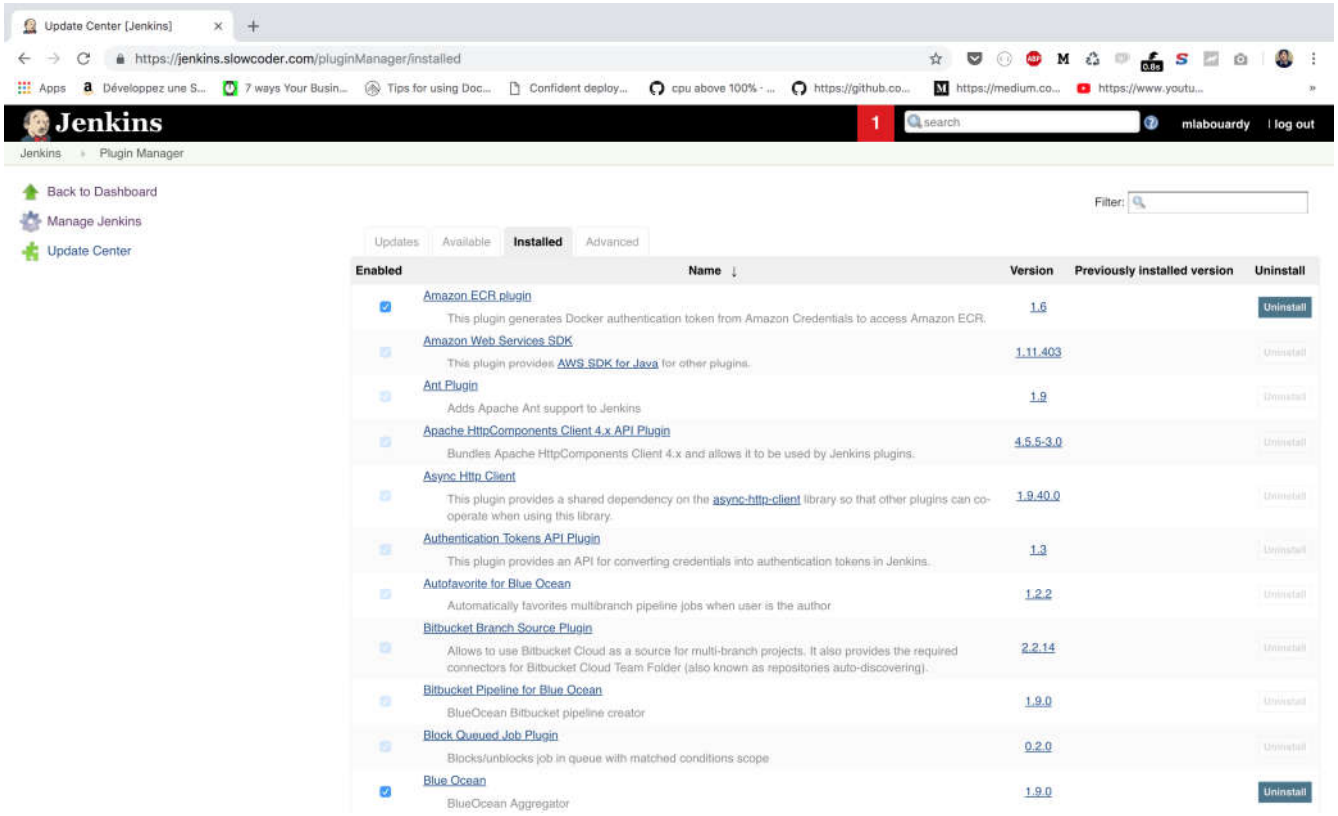


If you click on "**Credentials**" from the navigation pane, a set of credentials should be created out of the box:

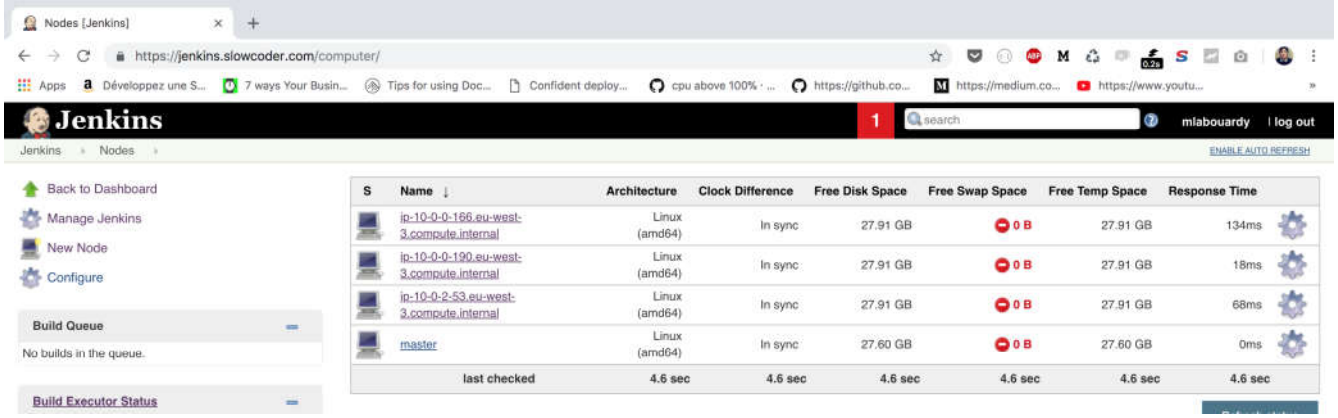


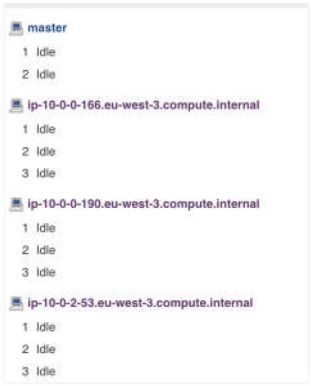


The same goes for "**Plugins**", a list of needed packages will be installed also:

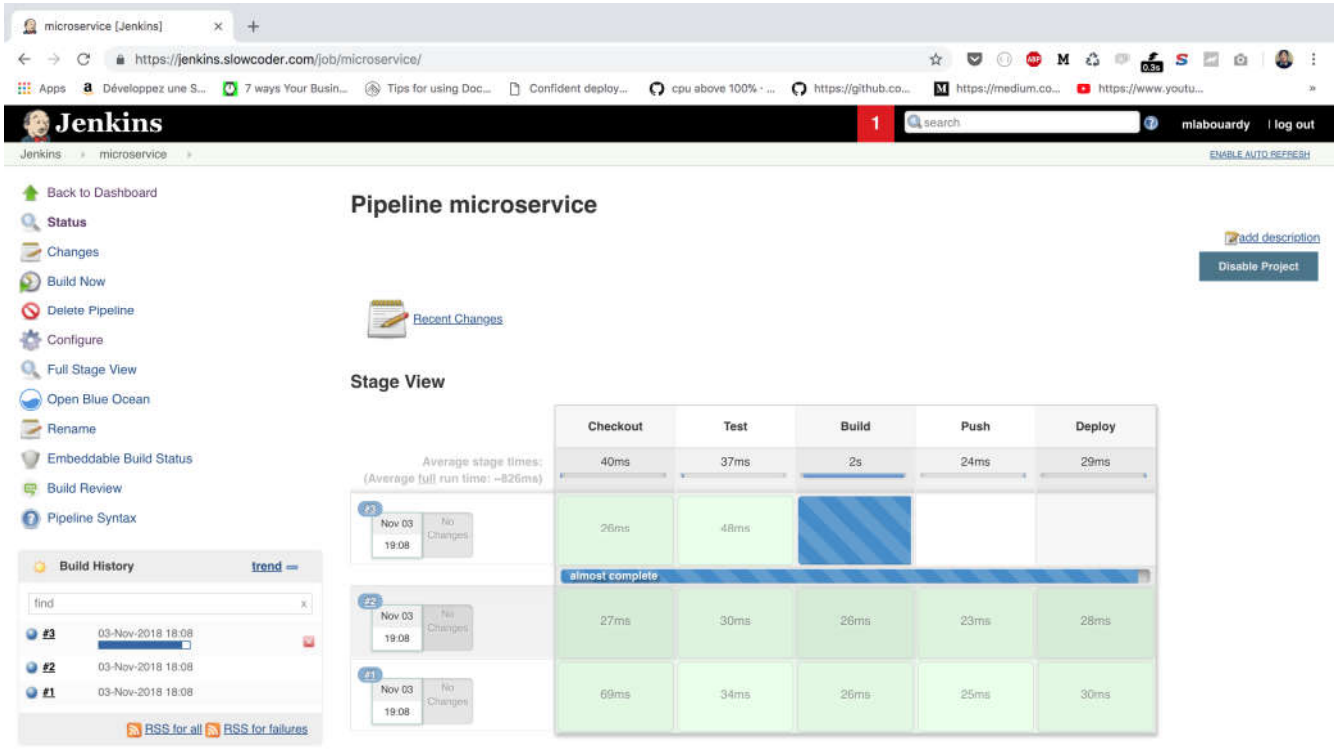


Once the Autoscaling group finished creating the EC2 instances, the instances will join the cluster automatically as you can see in the following screenshot:





You should now be ready to create your own CI/CD pipeline!



You can take this further and build a dynamic dashboard in your favorite visualization tool like *Grafana* to monitor your cluster resource usage based on the metrics collected by the agent installed on each EC2 instance: