Kubernetes Management Design Patterns

With Docker, CoreOS Linux, and Other Platforms

—

Deepak Vohra

Apress®

Kubernetes Management Design Patterns

With Docker, CoreOS Linux, and Other Platforms

Deepak Vohra

Apress®

Kubernetes Management Design Patterns: With Docker, CoreOS Linux, and Other Platforms

Deepak Vohra
White Rock, British Columbia
Canada

ISBN-13 (pbk): 978-1-4842-2597-4 ISBN-13 (electronic): 978-1-4842-2598-1
DOI 10.1007/978-1-4842-2598-1

Library of Congress Control Number: 2017930815

Cover image designed by Freepik.
Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Development Editor: Matthew Moodie
Technical Reviewer: Massimo Nardone
Coordinating Editor: Mark Powers
Copy Editor: James A. Compton
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/us/services/rights-permission.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers for download or cloning at Github via the book's product page, located at www.apress.com/9781484225974. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Contents at a Glance

Contents

About the Author

Deepak Vohra is a consultant and a principal member of the NuBean software company. Deepak is a Sun-certified Java programmer and Web component developer. He has worked in the fields of XML, Java programming, and Java EE for over seven years. Deepak is the coauthor of *Pro XML Development with Java Technology* (Apress, 2006). Deepak is also the author of *JDBC 4.0* and *Oracle JDeveloper for J2EE Development, Processing XML Documents with Oracle JDeveloper 11g, EJB 3.0 Database Persistence with Oracle Fusion Middleware 11g*, and *Java EE Development in Eclipse IDE* (Packt Publishing). He also served as the technical reviewer on *WebLogic: The Definitive Guide (O'Reilly Media, 2004)* and *Ruby Programming for the Absolute Beginner* (Cengage Learning PTR, 2007).

About the Technical Reviewer

Massimo Nardone has more than 22 years of experiences in Security, Web/Mobile development, Cloud and IT Architecture. His true IT passions are security and Android.

He has been programming and teaching how to program with Android, Perl, PHP, Java, VB, Python, C/C++ and MySQL for more than 20 years.

He holds a Master of Science degree in Computing Science from the University of Salerno, Italy.

He has worked as a Project Manager, Software Engineer, Research Engineer, Chief Security Architect, Information Security Manager, PCI/SCADA Auditor and Senior Lead IT Security/Cloud/SCADA Architect for many years.

Technical skills include: Security, Android, Cloud, Java, MySQL, Drupal, Cobol, Perl, Web and Mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, etc.

He currently works as Chief Information Security Office (CISO) for Cargotec Oyj.

He worked as visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (PKI, SIP, SAML and Proxy areas).

Massimo has reviewed more than 40 IT books for different publishing company and he is the coauthor of *Pro Android Games* (Apress, 2015).

Introduction

Docker was made available as open source in March 2013 and has become the most commonly used containerization platform. Kubernetes was open-sourced in June 2014 and has become the most widely used container cluster manager. The first stable version of CoreOS Linux was made available in July 2014 and since has become the most commonly used operating system for containers. My first book, *Kubernetes Microservices with Docker* (Apress, 2016), is an introduction to creating microservices with Kubernetes and Docker. This book, *Kubernetes Management Design Patterns,* takes container cluster management to the next level and discusses all or most aspects of administering and configuring Kubernetes on CoreOS and applying suitable design patterns such as ConfigMaps, autoscaling, resource quotas, and high availability. Kubernetes is a cluster manager for Docker and rkt containers, but this book discusses Kubernetes in the context of Docker only. A cluster manager for Docker containers is needed because the Docker engine by itself lacks some functionality, such as the ability to scale a cluster of containers, schedule pods on nodes, or mount a certain type of storage (such as an AWS Volume or Github repo) as volumes. Docker Engine 1.12 integrates the Docker Swarm cluster manager and Docker Swarm does overcome some of the earlier limitations of Docker by providing replication, load balancing, fault tolerance, and service discovery, but Kubernetes provides some features suitable for developing object-oriented applications. The Pod abstraction is the atomic unit of deployment in Kubernetes. A Pod may consist of one or more containers. Co-locating containers has several advantages as containers in a Pod share the same networking and filesystem and run on the same node. Docker Swarm does not support autoscaling directly. While Docker Swarm is Docker native, Kubernetes is more production-ready having been used in production at Google for more than 15 years.

Kubernetes Design Patterns

> *A software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design.*
>
> Wikepedia

A Docker image includes instructions to package all the required software and dependencies, set the environment variables, and run commands, and it is a reusable encapsulation of software for modular design. The atomic unit of modular container service in Kubernetes is a *pod*, which is a group of containers with a common filesystem and networking. The Kubernetes pod abstraction enables design patterns for containerized applications similar to object oriented design patterns. Pod, service, replication controller, deployment, and ConfigMap are all types of Kubernetes objects. Further, because containers interact with each other over HTTP, making use of a commonly available data format such as JSON, Kubernetes design patterns are language and platform independent. Containers provide some of the same benefits as software objects such as modularity or packaging, abstraction and reuse. Kubernetes has described three classes or types of patterns.

- Management design patterns
- Patterns involving multiple cooperating containers running on the same node
- Patterns involving containers running across multiple nodes

Some of the benefits of modular containers are as follows:

- The container boundary is an encapsulation or abstraction boundary that can be used to build modular, reusable components.
- The reusable containers may be shared between different applications and agile developer teams.
- Containers speed application development.
- Containers are suitable for agile team development.
- Containers can be used to encapsulate a best design or implementation.
- Containers provide separation of concerns

The design patterns are introduced in the publication Design Patterns For Container-Based Distributed Systems, by Brendan Burns and David Oppenheimer (`https://www.usenix.org/node/196347`). In this book we shall be using some of these and other design patterns.

Kubernetes Architecture

A Kubernetes cluster consists of a single *master node* (unless a high-availability master is used, which is not the default) and one or more *worker nodes* with Docker installed on each node. The following components run on each master node:

- *etcd* to store the persistent state of the master including all configuration data. A high-availability etcd cluster can also be used.
- An *API Server* to serve up the Kubernetes REST API for Kubernetes objects (pods, services, replication controllers, and others).
- *Scheduler* to bind unassigned pods on nodes.
- *Controller manager* performs all cluster level operations such as create and update service endpoints, discover, manage and monitor nodes. The replication controller is used to scale pods in a cluster.

The following components are run on each worker node:

- *kubelet* to manage the pods (including containers), Docker images, and volumes. The kubelet is managed from the API Server on the master node.
- *kube-proxy* is a network proxy and load balancer to serve up services.

The Kubernetes architecture is shown in Figure I-1.

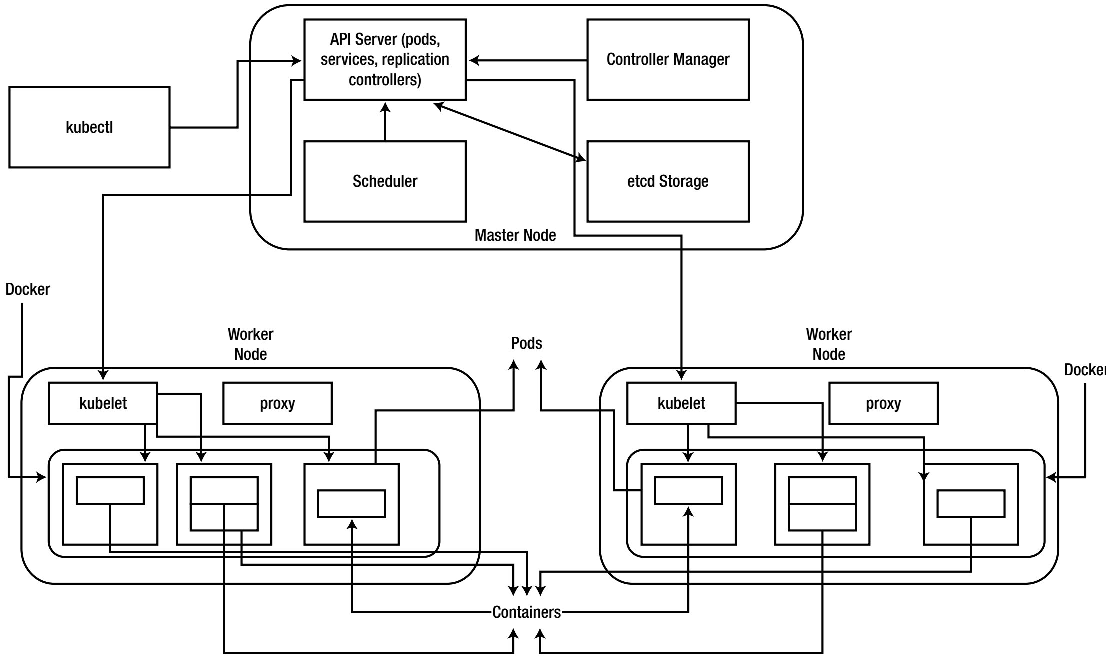


Figure I-1. *Kubernetes Architecture*

Why CoreOS?

CoreOS is the most widely used Linux OS designed for containers, not just Docker containers but also rkt (an implementation of the APP Container spec) containers. Docker and rkt are pre-installed on CoreOS out-of-the-box. CoreOS supports most cloud providers including Amazon Web Services (AWS) Elastic Compute Cloud (EC2), Google Cloud Platform, and virtualization platforms such as VMWare and VirtualBox. CoreOS provides Cloud-Config for declaratively configuring for OS items such as network configuration (flannel), storage (etcd), and user accounts. CoreOS provides a production-level infrastructure for containerized applications including automation, security and scalability. CoreOS has been leading the drive for container industry standards and in fact founded appc. CoreOS is not only the most widely used operating system for containers but also the most advanced container registry, Quay. CoreOS provides server security with Distributed Trusted Computing. CoreOS also provides Tectonic Enterprise for enterprise-level workloads without operational overhead and an out-of-the-box Kubernetes cluster and a user-friendly dashboard.

Chapter Description

In Chapter 1 we shall install Kubernetes on Amazon Web Services (AWS), create a sample deployment and service, and subsequently invoke the service. Kubernetes installation on AWS requires almost no configuration to spin-up a multi-node cluster.

In Chapter 2 we shall install Kubernetes on CoreOS, which is the main platform we shall use for most of the chapters. We'll first create an AWS EC2 instance from Amazon Linux AMI, which has the AWS Command Line Interface (CLI) preinstalled. We'll then SSH log in to the EC2 instance and install Kube-aws. Then we will launch a CloudFormation for a Kubernetes cluster with one controller node and three worker nodes and SSH log in to the controller instance and install kubectl binaries to access the API server.

In Chapter 3 we shall discuss Google Cloud Platform for Kubernetes. First, create a project and a VM instance. Subsequently connect to the VM instance to create a Kubernetes cluster and test a sample application.

In Chapter 4 we shall use multiple zones to create an AWS CloudFormation for a Kubernetes cluster.

Chapter 5 introduces the Tectonic Console for managing Kubernetes applications deployed on CoreOS.

Chapter 6 is on volumes. We demonstrate using volumes with two types of volumes: awsElasticBlockStore volume and gitRepo volume.

Chapter 7 is on using services. We shall create sample services for three kinds of services supported by Kubernetes: ClusterIP, NodePort and LoadBalancer.

In Chapter 8 we shall discuss rolling updates. A rolling update is the mechanism by which a running replication controller can be updated to a newer image or specification while it is running.

In Chapter 9 we introduce the scheduling policy used by Kubernetes to schedule pods on nodes. We discuss the various options including using a NodeSelector, and setting node affinity.

Chapter 10 is on allocating compute resources to applications. The two supported compute resources are CPU and memory. We shall discuss setting resource requests and limits and also how Kubernetes provides a quality of service by guaranteeing a preset level of resources.

Chapter 11 is on ConfigMaps, which are maps of configuration properties that may be used in pods and replication controller definition files to set environment variables, command arguments and such.

Chapter 12 is on setting resource quotas on namespaces for constraining resource usage in a namespace. Resource quotas are useful in team development (different teams have different requirements) and different phases of application which have different resource requirements such as development, testing, and production.

Chapter 13 is on autoscaling, which is suitable for production workloads that can fluctuate. Autoscaling of a deployment, replica set, or replication controller scales the number of pods in a cluster automatically when the load fluctuates.

Chapter 14 is on logging. The default logger is discussed in addition to cluster-level logging using Elasticsearch, Fluentd, and Kibana.

In Chapter 15 OpenShift, a PaaS platform for Kubernetes, is discussed to create a high availability master Kubernetes cluster using Ansible. Ansible is an automation platform for application deployment, configuration management, and orchestration.

In Chapter 16 a high availability web site is developed using AWS Route 53 for DNS failover.

PART I

Platforms

CHAPTER 1

Kubernetes on AWS

Kubernetes is a cluster manager for Docker (and rkt) containers. The Introduction outlines its basic architecture and relationship to CoreOS and Amazon Web Services (AWS). In this chapter we'll spin up a basic cluster without configuration.

Note *Kubernetes Microservices with Docker* (Apress, 2016) covers installing Kubernetes on single-node and multi-node clusters.

Problem

Installing Kubernetes by installing its individual components (Docker, Flannel, Kubelet, and Service Proxy) separately is an involved process that requires many commands to be run and files to be configured.

Solution

AWS provides a legacy tool called `kube-up.sh` to spin up a Kubernetes cluster without requiring any configuration. Only an AWS account, the AWS Command Line Interface (CLI), and access to the AWS APIs are required. Kubernetes and other tools such as Elasticsearch (used to index and store logs), Heapster (used to analyze compute resource usage), Kibana (a GUI dashboard used to view the logs), KubeDNS (used to resolve DNS names for services), Kubernetes-dashboard, Grafana (used for metrics visualization), and InfluxDB are all installed with a single command.

Overview

In this chapter we will create a multi-node cluster (consisting of one master and multiple minions) on Amazon Elastic Compute Cloud (EC2) using the AWS Command Line Interface. The stages are as follows:

Setting the Environment

Starting a Cluster

Testing the Cluster

Configuring the Cluster

Stopping the Cluster

D. Vohra, *Kubernetes Management Design Patterns*, DOI 10.1007/978-1-4842-2598-1_1

Setting the Environment

Because we're using Amazon EC2, an AWS account is required. Also, to configure AWS you need to obtain security credentials. Select Security Credentials for a user account. In the Your Security Credentials screen, select the Access Keys node and click Create New Access Key as shown in Figure 1-1 to create a new access key.

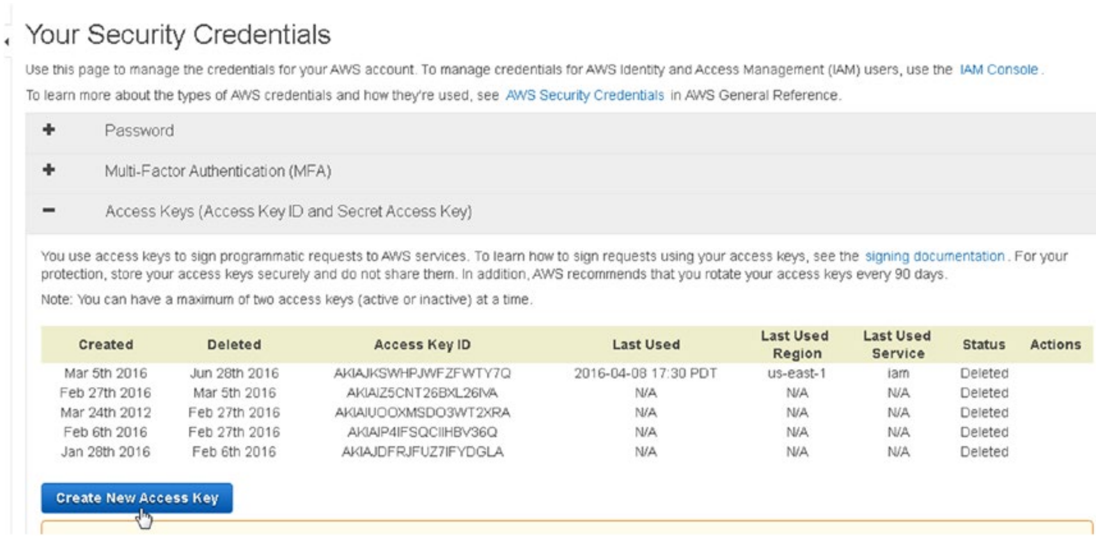


Figure 1-1. Creating a new access key

A new security access key is created and the Access Key ID and Secret Access Key are listed.

Copy the Access Key ID and Secret Access Key to be used later to configure AWS. The Access Key ID and Secret Access Key will be different for different users.

```
AWS_ACCESS_KEY_ID   AKIAISQVxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY   VuJD5gDxxxxxxxxxxxxxxxxxxxx
```

Because the AWS Command Line Interface is required, create an EC2 instance of the Amazon Linux Amazon Machine Image (AMI), which has the AWS CLI preinstalled. Click on Launch Instance as shown in Figure 1-2 to create a new instance.

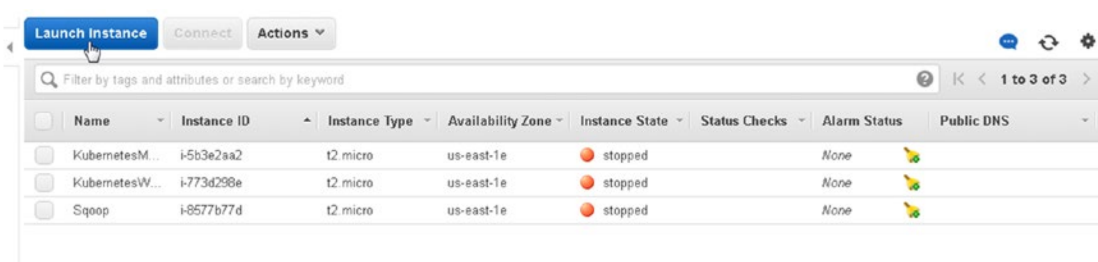


Figure 1-2. Launching an EC2 instance

In the next screen, select the Amazon Linux AMI (64 bit) as shown in Figure 1-3.

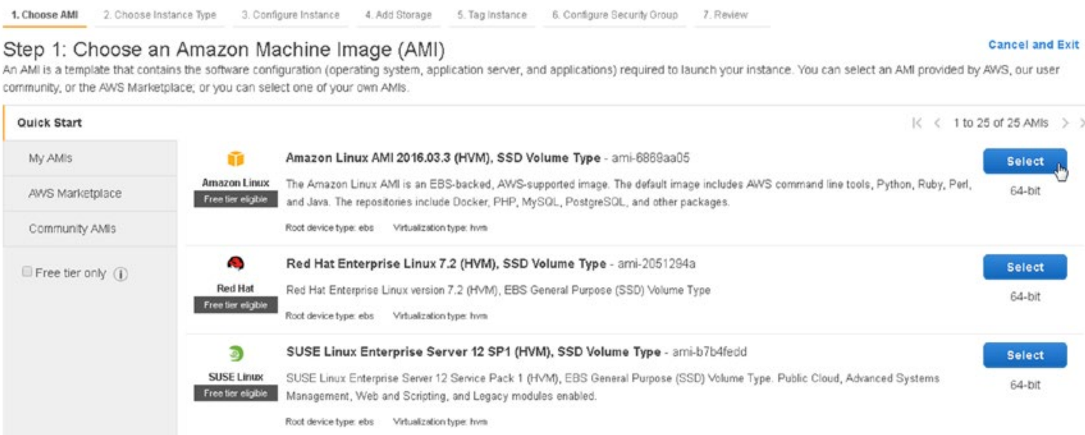


Figure 1-3. *Selecting Amazon Linux AMI*

For the Instance Type, select a relatively large Instance size (m4.xlarge) as shown in Figure 1-4, because the default (Free Tier) micro size may not provide sufficient memory or disk space to install Kubernetes. Some of the instance types such as m4.xlarge may only be launched in a virtual private cloud (VPC). When you are ready, click Next:Configure Instance Details.

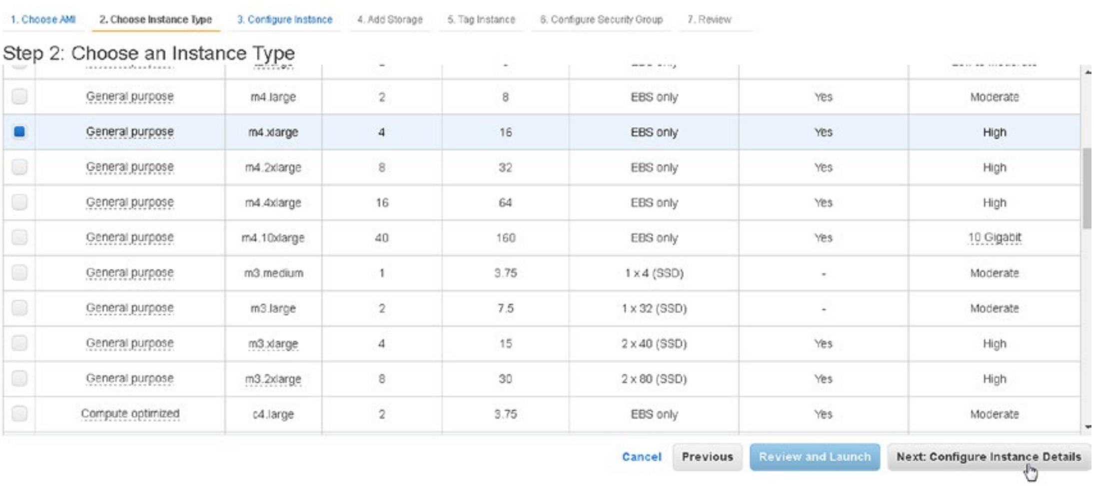


Figure 1-4. *Choosing an instance type*

Specify the instance details such as Network VPC and Subnet as shown in Figure 1-5. When finished, click Next: Add Storage.

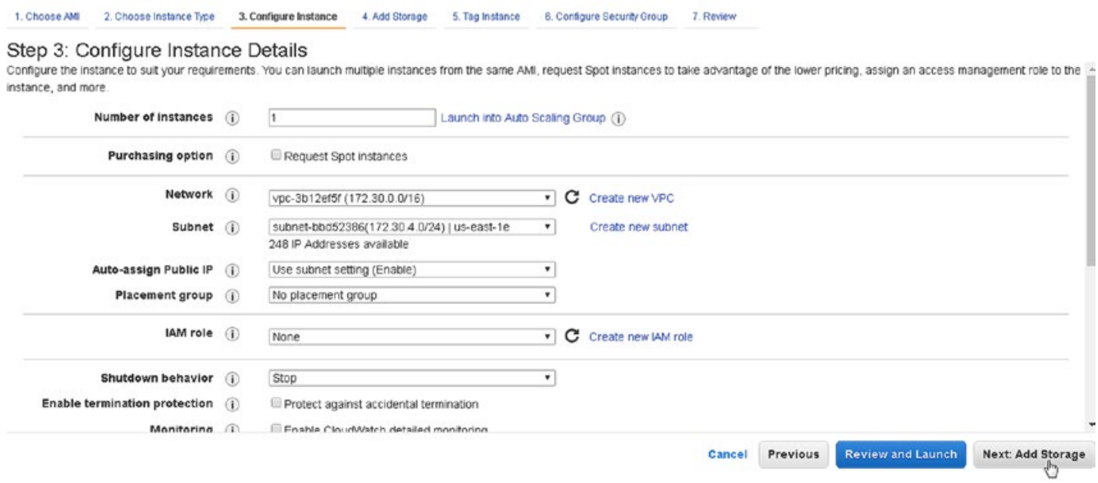


Figure 1-5. Configuring instance details

A new EC2 instance is created. Obtain the Public DNS for the instance as shown in Figure 1-6.

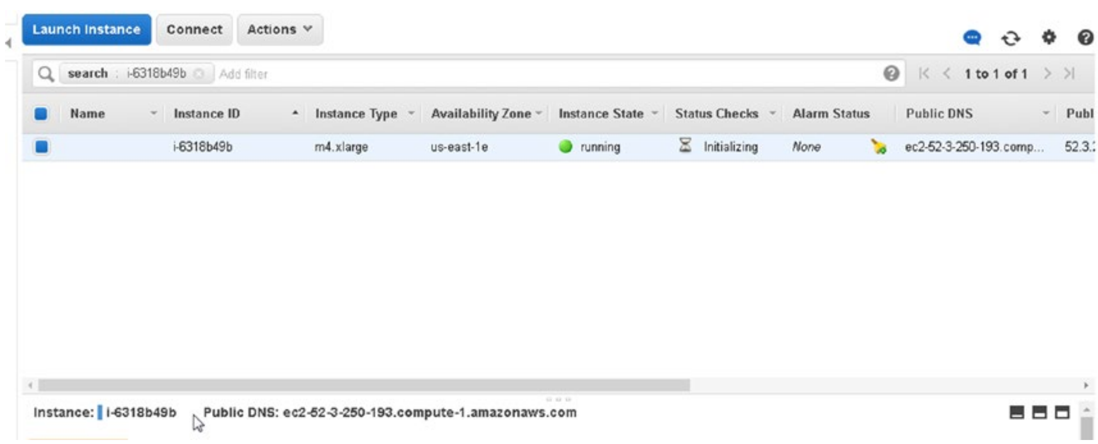


Figure 1-6. The public DNS

Using the private key that was specified when the instance was created, SSH log in to the instance:

```
ssh -i "docker.pem"  ec2-user@ec2-52-3-250-193.compute-1.amazonaws.com
```

The Amazon Linux command prompt is displayed as shown in Figure 1-7.

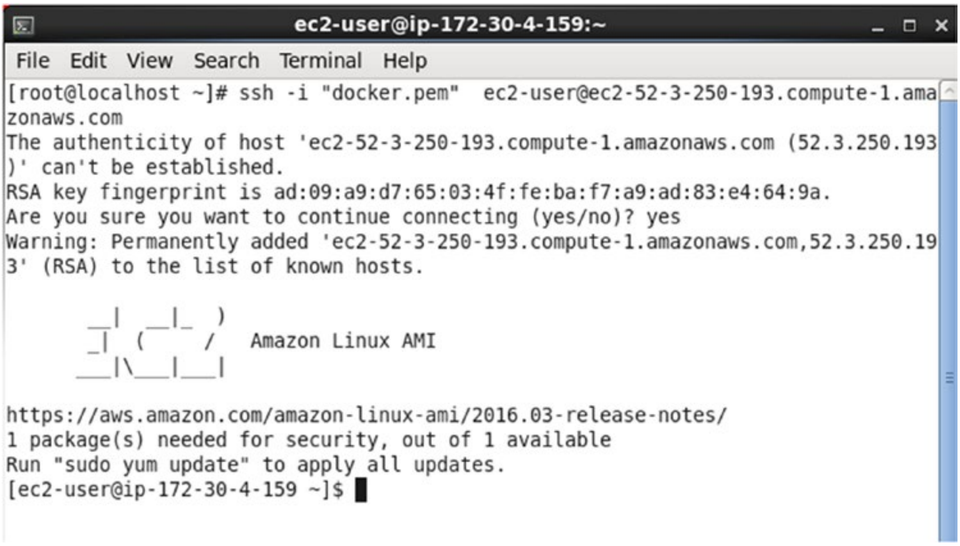


Figure 1-7. *Amazon Linux AMI command prompt*

Configuring AWS

When a Kubernetes cluster is started on AWS EC2, a new VPC is created for the master and minion nodes. The number of VPCs that may be created in an AWS account has a limit, which can vary for different users. Before starting the cluster, delete the VPCs that are not being used so that the limit is not reached when a new VPC is created. To begin, select VPC in the AWS Services as shown in Figure 1-8.

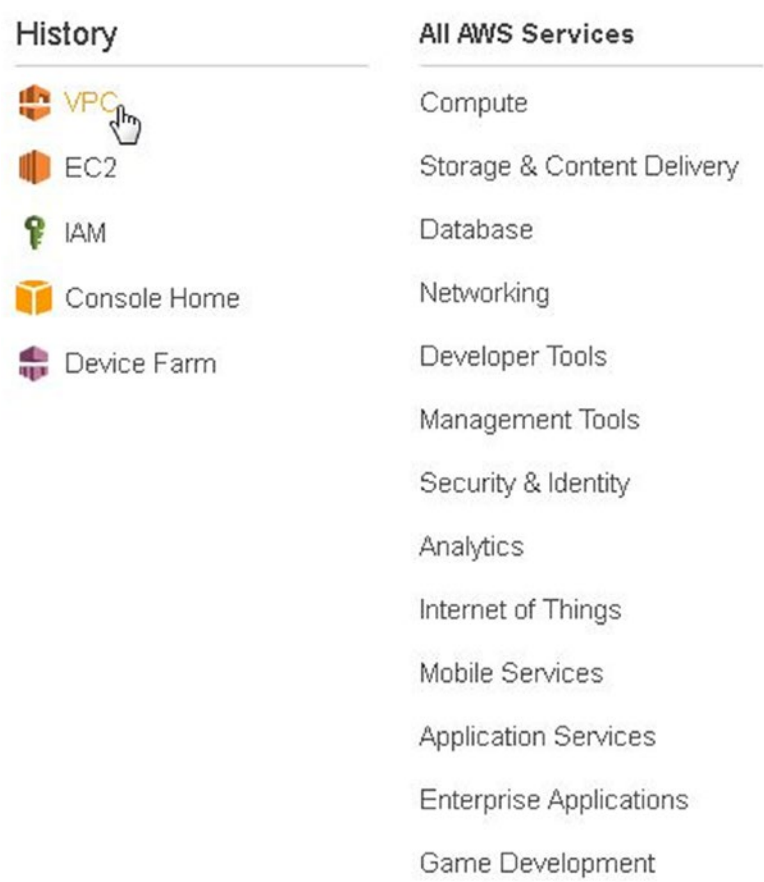


Figure 1-8. *Selecting the VPC console*

Click on Start VPC Wizard as shown in Figure 1-9 to list and delete VPCs if required.

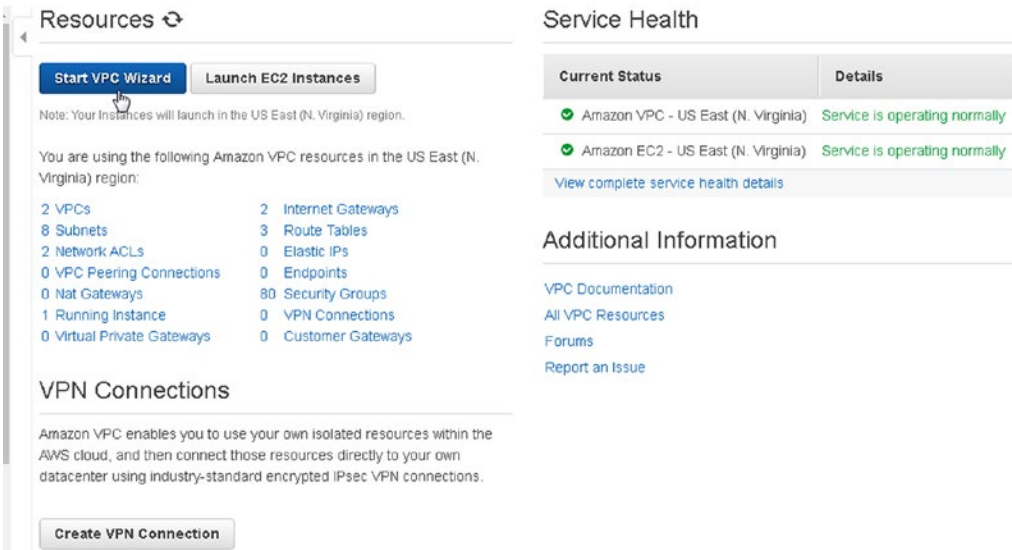


Figure 1-9. *Starting the VPC Wizard*

The VPCs already available are listed as shown in Figure 1-10.

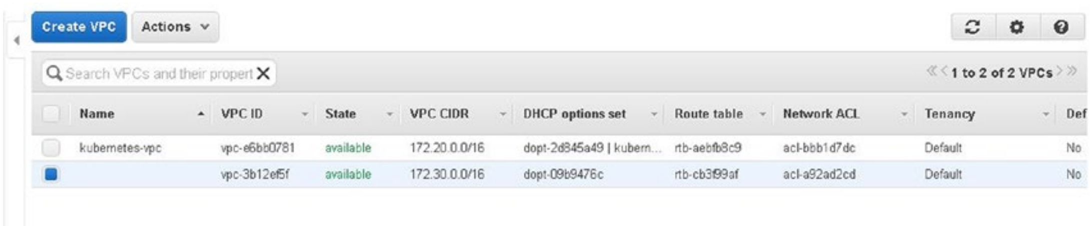


Figure 1-10. *Available VPCs*

To delete a VPC, select the VPC and click Actions ➤ Delete VPC, as shown in Figure 1-11.

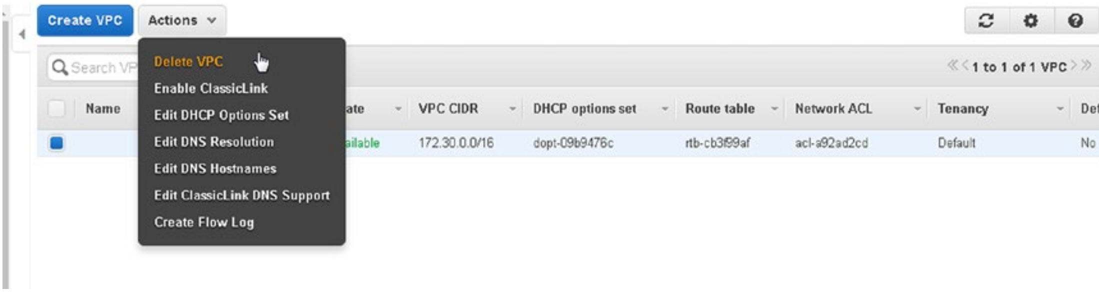


Figure 1-11. *Selecting Actions ➤ Delete VPC*

In the confirmation screen that appears, click Yes, Delete. If the VPC is not associated with any instance, the VPC should start to be deleted as shown in Figure 1-12.

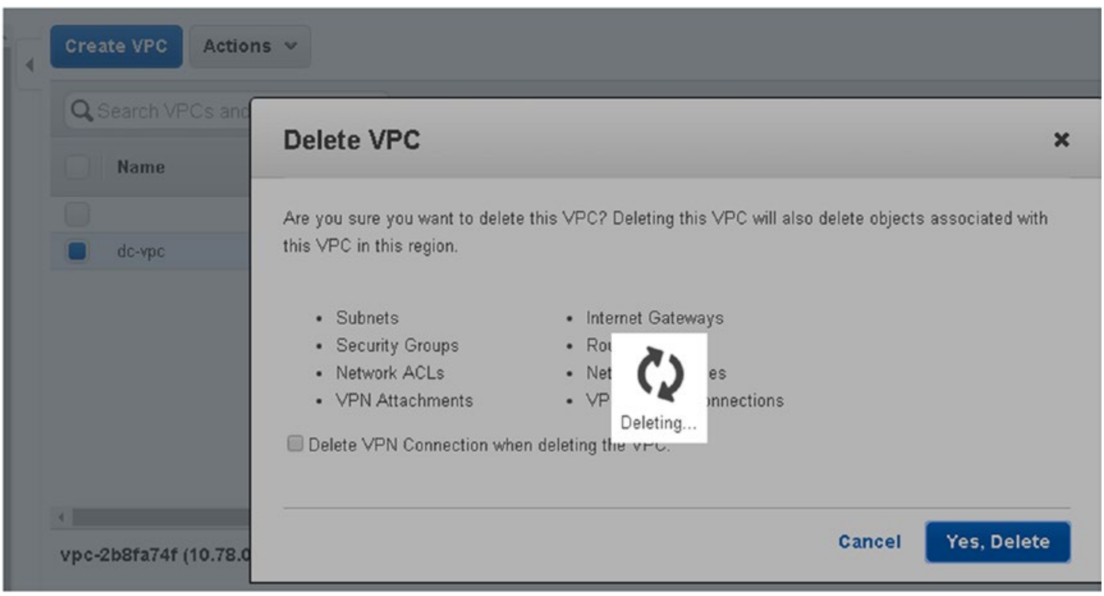


Figure 1-12. *Deleting a VPC*

If a VPC is associated with any instance, then it is not deletable and the Yes, Delete button is unavailable, as shown in Figure 1-13.

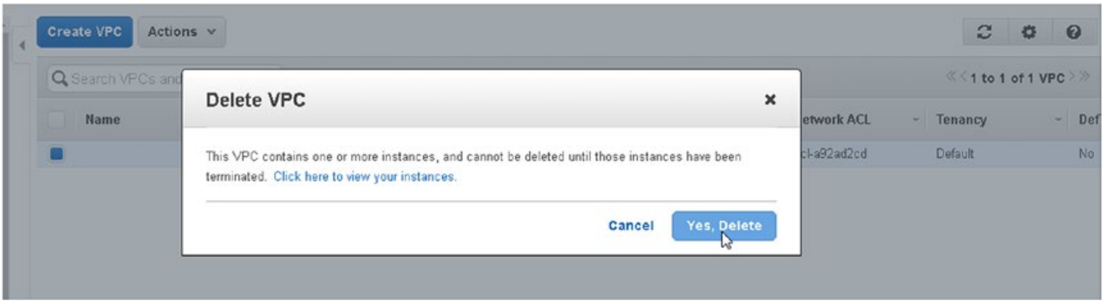


Figure 1-13. *The message for a nondeletable VPC*

Next, configure AWS on the Amazon Linux instance using the following command:

```
aws configure
```

When prompted, specify the AWS Access Key ID and AWS Access Key. Also specify the default region name (`us-east-1`) and the default output format (`json`) as shown in Figure 1-14.

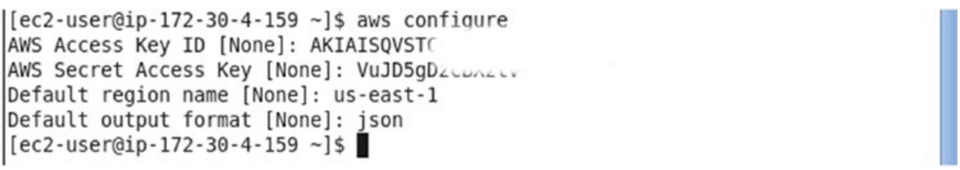


Figure 1-14. *Configuring AWS*

Starting the Kubernetes Cluster

Now that you have configured AWS, run the following command to install Kubernetes:

```
export KUBERNETES_PROVIDER=aws; wget -q -O - https://get.k8s.io | bash
```

This command starts the Kubernetes installation process as shown in Figure 1-15.

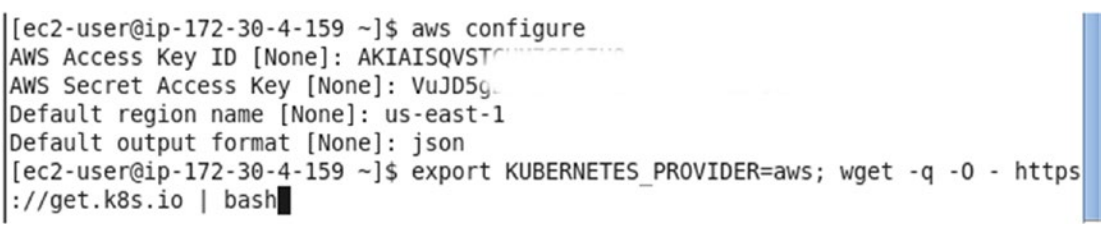
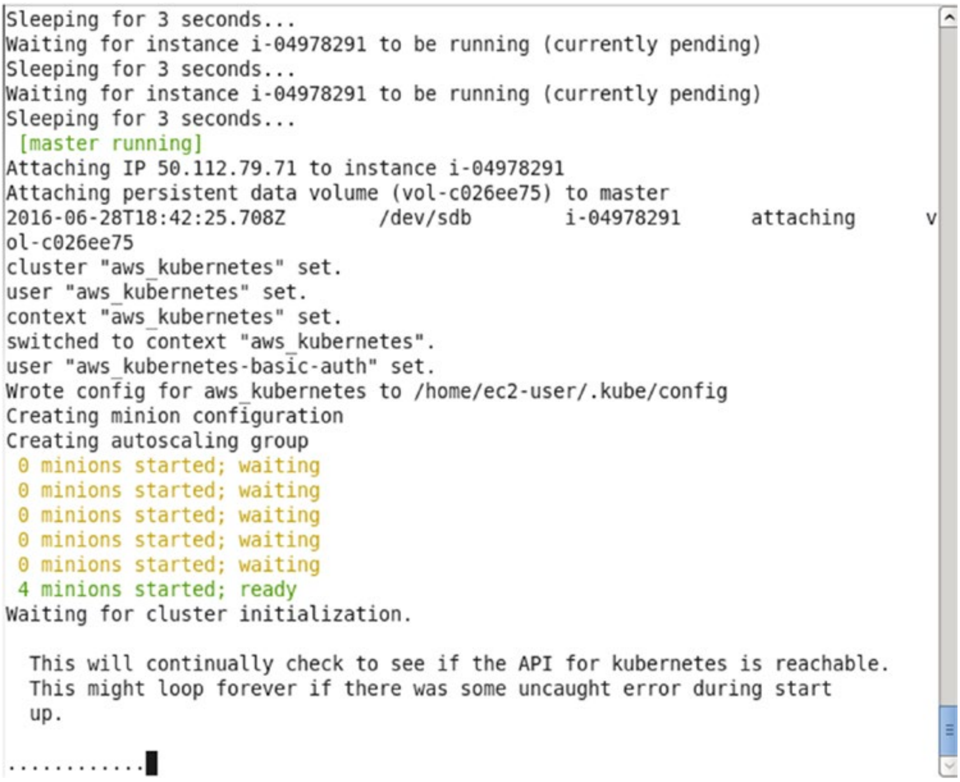


Figure 1-15. *Installing Kubernetes*

The preceding command invokes the `cluster/kube-up.sh` script, which further invokes the `cluster/aws/util.sh` script using the configuration specified in the `cluster/aws/config-default.sh` script. One master and four minions are started on Debian 8 (jessie) as shown in Figure 1-16. The cluster initialization is started subsequently.



```
Sleeping for 3 seconds...
Waiting for instance i-04978291 to be running (currently pending)
Sleeping for 3 seconds...
Waiting for instance i-04978291 to be running (currently pending)
Sleeping for 3 seconds...
 [master running]
Attaching IP 50.112.79.71 to instance i-04978291
Attaching persistent data volume (vol-c026ee75) to master
2016-06-28T18:42:25.708Z        /dev/sdb        i-04978291      attaching       v
ol-c026ee75
cluster "aws_kubernetes" set.
user "aws_kubernetes" set.
context "aws_kubernetes" set.
switched to context "aws_kubernetes".
user "aws_kubernetes-basic-auth" set.
Wrote config for aws_kubernetes to /home/ec2-user/.kube/config
Creating minion configuration
Creating autoscaling group
 0 minions started; waiting
 0 minions started; waiting
 0 minions started; waiting
 0 minions started; waiting
 0 minions started; waiting
 4 minions started; ready
Waiting for cluster initialization.

  This will continually check to see if the API for kubernetes is reachable.
  This might loop forever if there was some uncaught error during start
  up.

............
```

Figure 1-16. *Starting master and minions*

The cluster is started and validated, and the components installed are listed. The URLs at which the Kubernetes master, Elasticsearch, Heapster, and other services are running are listed as shown in Figure 1-17. The directory path at which the Kubernetes binaries are installed is also listed.