Kubernetes the Hard Way

Course Features and Tools

Kubernetes the Hard Way

Course Introduction

Kubernetes the Hard Way

This course is adapted from the original open-source Kubernetes the Hard Way guide created by Google's Kelsey Hightower.

You can find the original Kubernetes the Hard Way guide here:
https://github.com/kelseyhightower/kubernetes-the-hard-way

Kubernetes the Hard Way guides you through the process of setting up a Kubernetes cluster manually, without automated scripts or installers.

This course is for anyone who wants a deeper understanding of the inner workings of Kubernetes.

Kubernetes the Hard Way

Our version of Kubernetes the Hard Way closely follows the original guide.

However, it has been adapted for Linux Academy, and a few changes have been made to make it work on the Linux Academy platform.

But for the most part, the approach is the same. Most of the commands you will be using are exactly the same as the ones used in the original open-source guide!

The primary difference between the original guide and this course is that the original guide is built on Google Cloud Platform, whereas this course uses Linux Academy cloud servers.

Prerequisites

To get the most out of this course, you should already be familiar with the following:

- The basic features of Kubernetes and how it is used.
    - If you are completely new to Kubernetes, check out the "Core Concepts" section of our Certified Kubernetes Administrator course.
- The Linux command line.

Course Structure

This course follows the structure of the original Kubernetes the Hard Way guide.

The course is divided into 13 sections, and each one guides you through a different section of the original guide.

You can follow along with the lessons and build your own Kubernetes cluster using our cloud servers.

Each section also contains learning activities that let you practice what you've learned in a controlled environment.

Kubernetes the Hard Way

What Does the Kubernetes Cluster Architecture Look Like?


Kubernetes the Hard Way
Setting Up Your Cloud Servers

Provisioning Servers

For this course, we will need a total of five cloud servers:

- 2 Kubernetes controllers
- 2 Kubernetes worker nodes
- 1 Kubernetes API load balancer

The operating system we will be using is Ubuntu 16.

Kubernetes the Hard Way

Client Tools

Client Tools

For this course, we'll need two client tools: cfssl and kubectl.

Check out the *Kubernetes the Hard Way* docs for more information on how to install these!

https://github.com/kelseyhightower/kubernetes-the-hard-way/blob/master/docs/02-client-tools.md

Kubernetes the Hard Way

Why Do We Need a CA and TLS Certificates?

The Certificate Authority and Certificates

In this section, we will be provisioning a certificate authority (CA).

We will then use the CA to generate several certificates.

Certificates are used to confirm (authenticate) identity. They are used to prove that you are who you say you are.

A Certificate Authority provides the ability to confirm that a certificate is valid. A certificate authority can be used to validate any certificate that was issued using that certificate authority.

Kubernetes uses certificates for a variety of security functions, and the different parts of our cluster will validate certificates using the certificate authority. In this section, we will generate all of these certificates and copy the necessary files to the servers that need them.

What Certificates Do We Need?

After we provision the certificate authority, we will need to generate certificates for the following:

- Client Certificates – These certificates provide client authentication for various users: admin, kube-controller-manager, kube-proxy, kube-scheduler, and the kubelet client on each worker node.
- Kubernetes API Server Certificate – This is the TLS certificate for the Kubernetes API.
- Service Account Key Pair – Kubernetes uses a certificate to sign service account tokens, so we need to provide a certificate for that purpose.

Kubernetes the Hard Way

Provisioning the Certificate Authority

Kubernetes the Hard Way

Generating Client Certificates


Kubernetes the Hard Way
Generating the Kubernetes API Server Certificate


Kubernetes the Hard Way
Generating the Service Account Key Pair


Kubernetes the Hard Way
Distributing the Certificate Files


Kubernetes the Hard Way
What Are Kubeconfigs and Why Do We Need Them?

Kubeconfigs

A Kubernetes configuration file, or kubeconfig, is a file that stores "information about clusters, users, namespaces, and authentication mechanisms." It contains the configuration data needed to connect to and interact with one or more Kubernetes clusters.

You can find more information about kubeconfigs in the Kubernetes documentation:
https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

Kubeconfigs contain information such as:

- The location of the cluster you want to connect to
- What user you want to authenticate as
- Data needed in order to authenticate, such as tokens or client certificates

You can even define multiple contexts in a kubeconfig file, allowing you to easily switch between multiple clusters.

Why Do We Need Kubeconfigs?

We use kubeconfigs to store the configuration data that will allow the many components of Kubernetes to connect to and interact with the Kubernetes cluster.

How will the kubelet service on one of our worker nodes know how to locate the Kubernetes API and authenticate with it? It will use a kubeconfig!

In the next lesson, we will generate the kubeconfigs that our cluster needs.

Kubernetes the Hard Way

Generating Kubeconfigs for the Cluster

How to Generate a Kubeconfig

Kubeconfigs can be generated using kubectl:

- Use kubectl config set-cluster to set up the configuration for the location of the cluster.
- Use kubectl config set-credentials to set the username and client certificate that will be used to authenticate.
- Use kubectl config set-context default to set up the default context.
- Use kubectl config use-context default to set the current context to the configuration we provided.

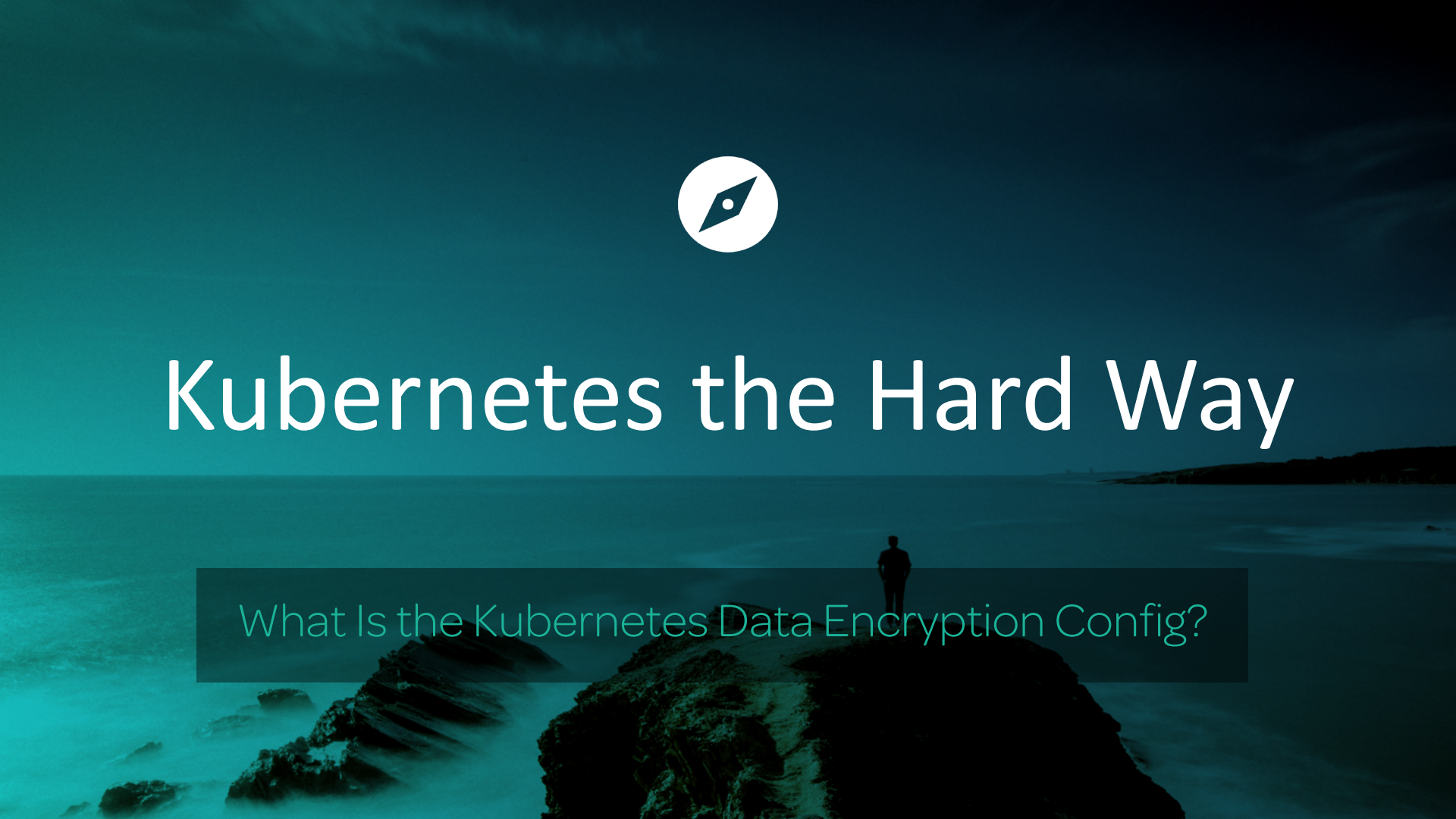
What Kubeconfigs Do We Need to Generate?

We will need several Kubeconfig files for various components of the Kubernetes cluster:

- Kubelet (one for each worker node)
- Kube-proxy
- Kube-controller-manager
- Kube-scheduler
- Admin

Kubernetes the Hard Way

Distributing the Kubeconfig Files



Kubernetes the Hard Way

What Is the Kubernetes Data Encryption Config?

Kubernetes Secret Encryption

Kubernetes supports the ability to encrypt secret data at rest.

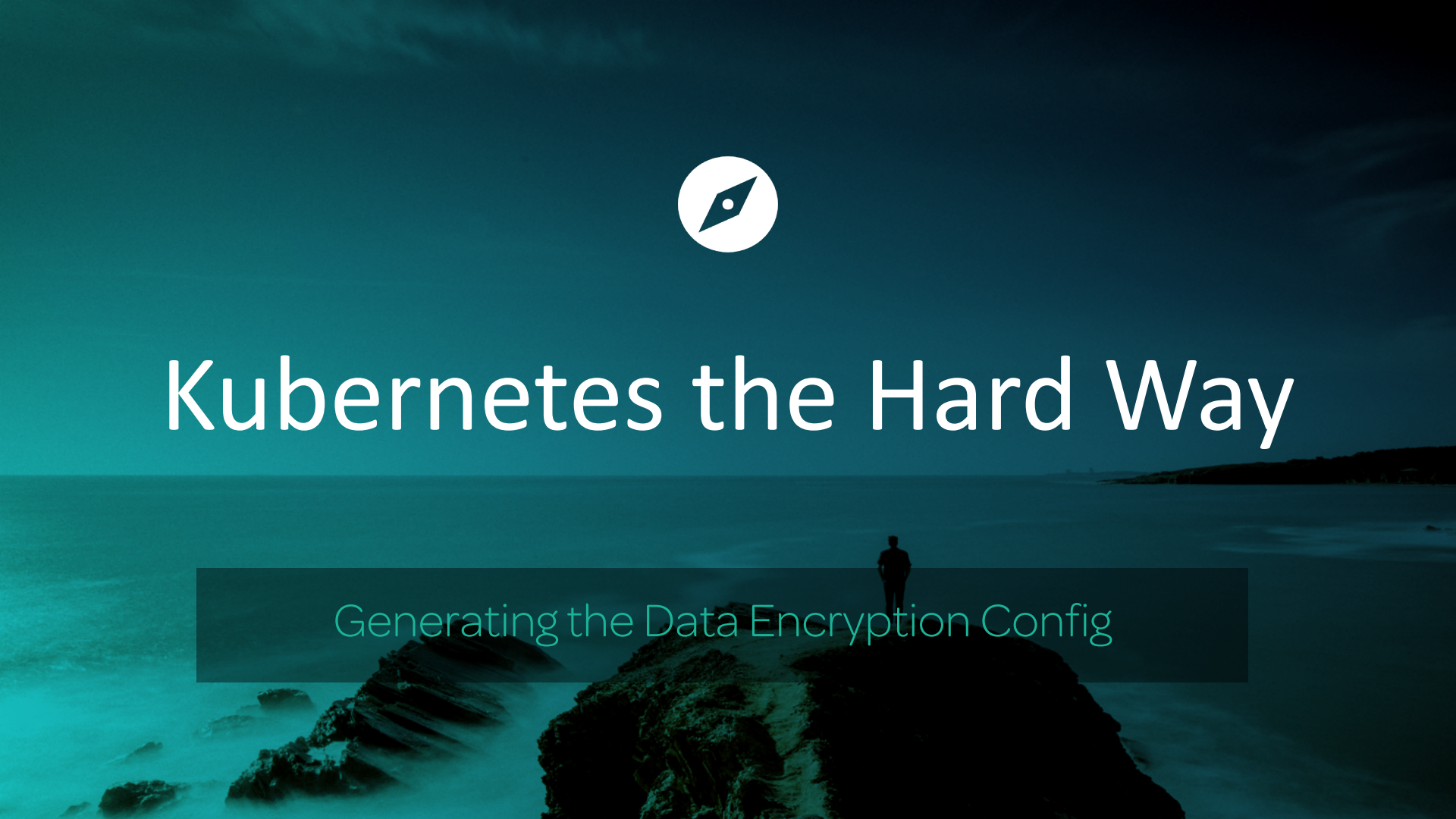
This means that secrets are encrypted so that they are never stored on disc in plain text.

This feature is important for security, but in order to use it we need to provide Kubernetes with an encryption key.

We will generate an encryption key and put it into a configuration file. We will then copy that file to our Kubernetes controller servers.

You can find more information on Kubernetes secret encryption in the official docs:

https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/



Kubernetes the Hard Way

Generating the Data Encryption Config


Kubernetes the Hard Way
What Is etcd?

What Is etcd?

"etcd is a distributed key value store that provides a reliable way to store data across a cluster of machines."

https://coreos.com/etcd/

etcd provides a way to store data across a distributed cluster of machines and make sure the data is synchronized across all machines.

You can find more information, as well as the etcd source code, in the etcd GitHub repository:

https://github.com/coreos/etcd

How Is etcd Used in Kubernetes?

Kubernetes uses etcd to store all of its internal data about cluster state.

This data needs to be stored, but it also needs to be reliably synchronized across all controller nodes in the cluster. etcd fulfills that purpose.

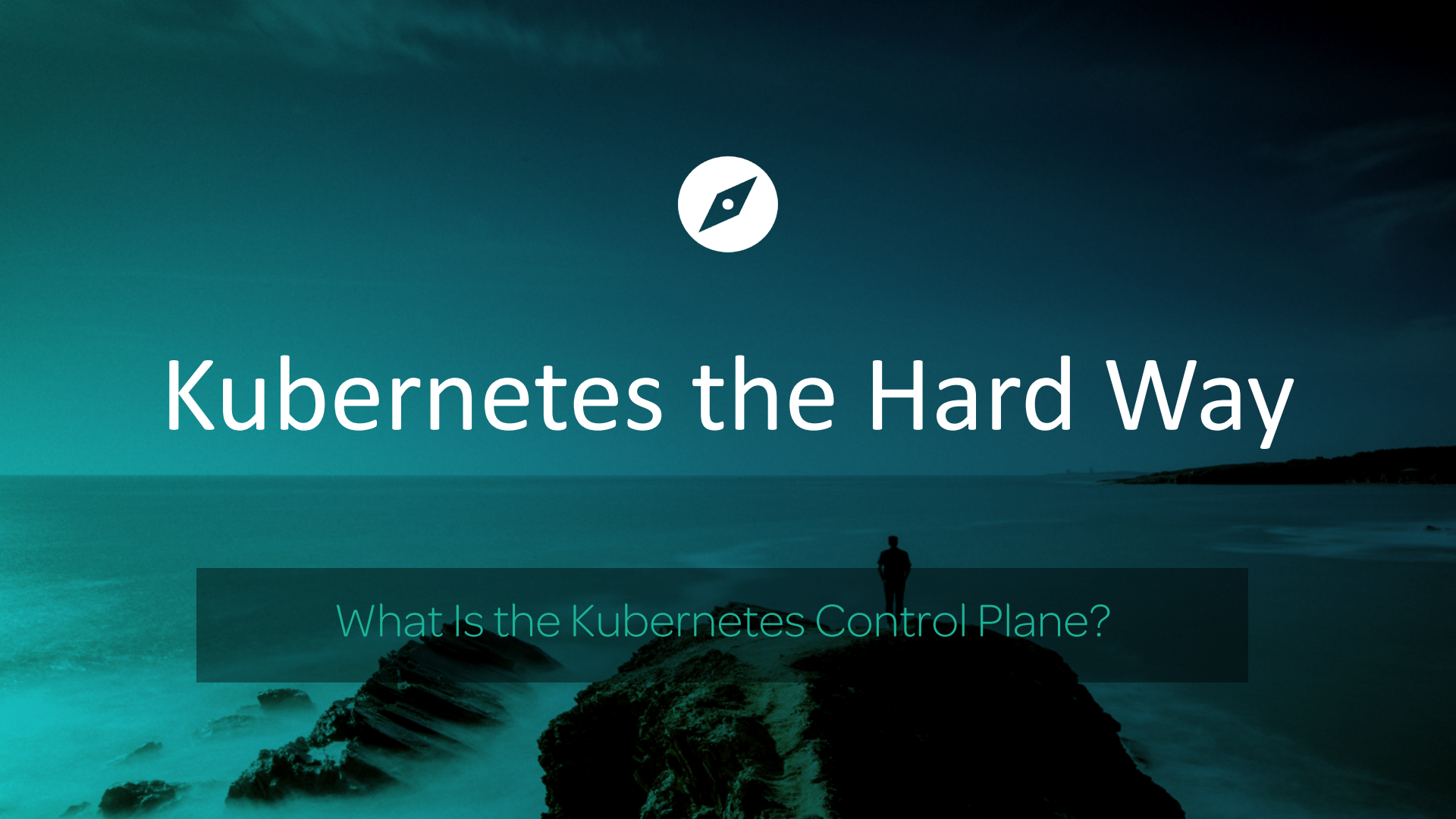
We will need to install etcd on each of our Kubernetes controller nodes and create an etcd cluster that includes all of those controller nodes.

You can find more information on managing an etcd cluster for Kubernetes here:

https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/

Kubernetes the Hard Way

Creating the etcd Cluster



Kubernetes the Hard Way

What Is the Kubernetes Control Plane?

What Is the Kubernetes Control Plane?

The Kubernetes control plane is a set of services that control the Kubernetes cluster.

Control plane components "make global decisions about the cluster (e.g., scheduling) and detect and respond to cluster events (e.g., starting up a new pod when a replication controller's *replicas* field is unsatisfied)."

You can find more information on the control plane in the official Kubernetes documentation:

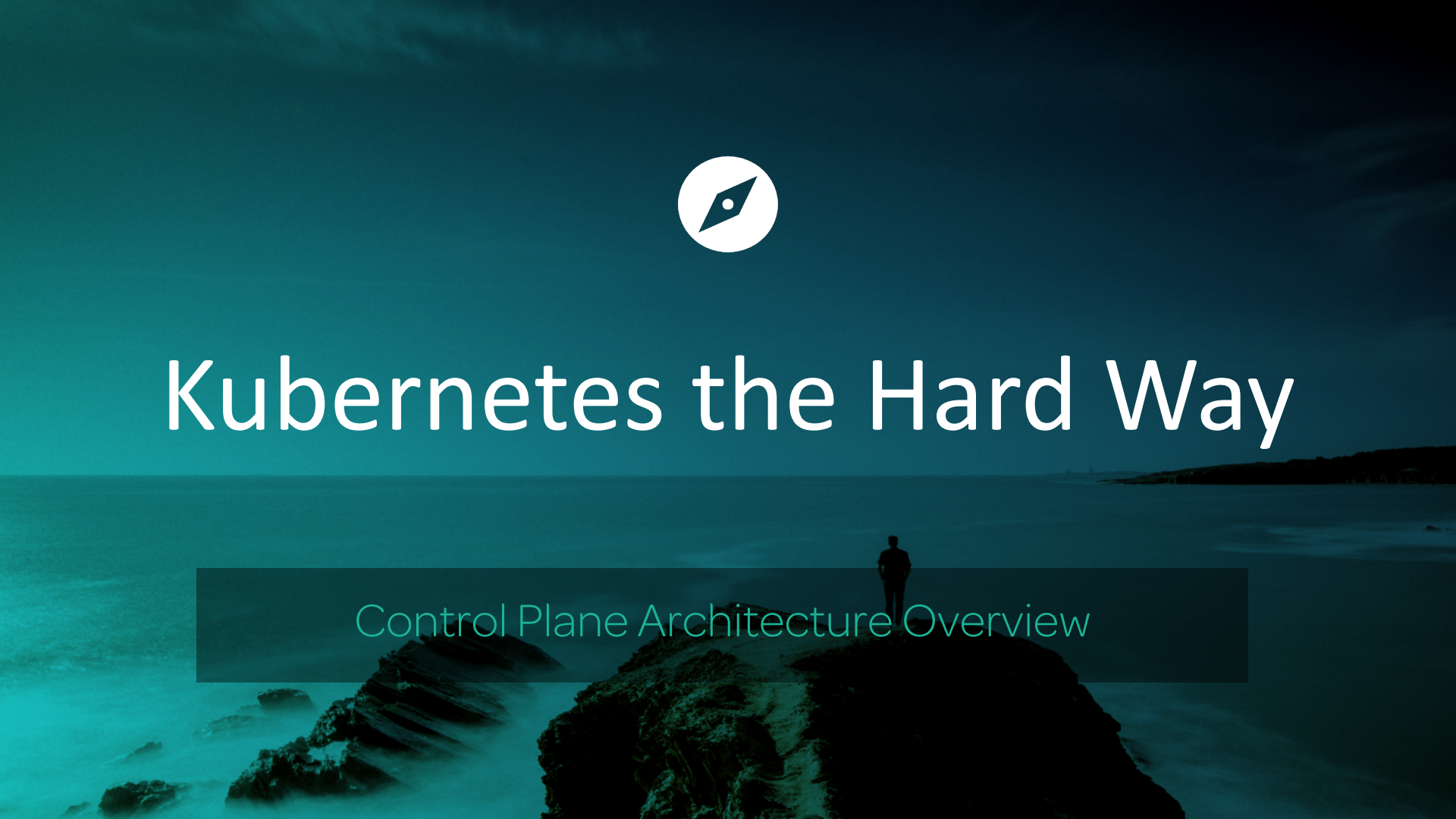
https://kubernetes.io/docs/concepts/overview/components/#master-components

Kubernetes Control Plane Components

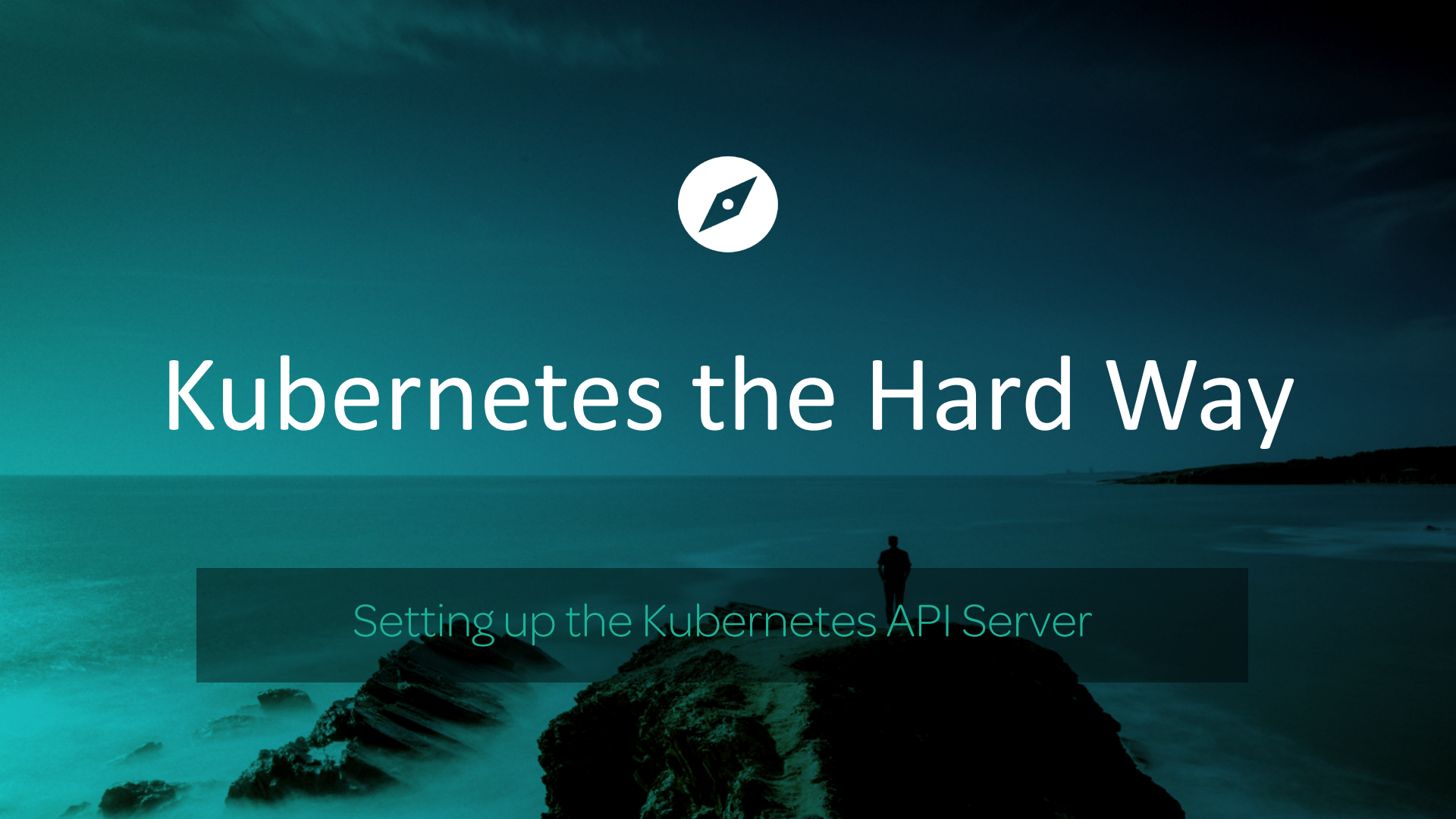
The Kubernetes control plane consist of the following components:

- kube-apiserver: Serves the Kubernetes API. This allows users to interact with the cluster.
- etcd: Kubernetes cluster datastore. (We've already installed this one.)
- kube-scheduler: Schedules pods on available worker nodes.
- kube-controller-manager: Runs a series of controllers that provide a wide range of functionality.
- cloud-controller-manager: Handles interaction with underlying cloud providers. (We won't use this one in the course.)

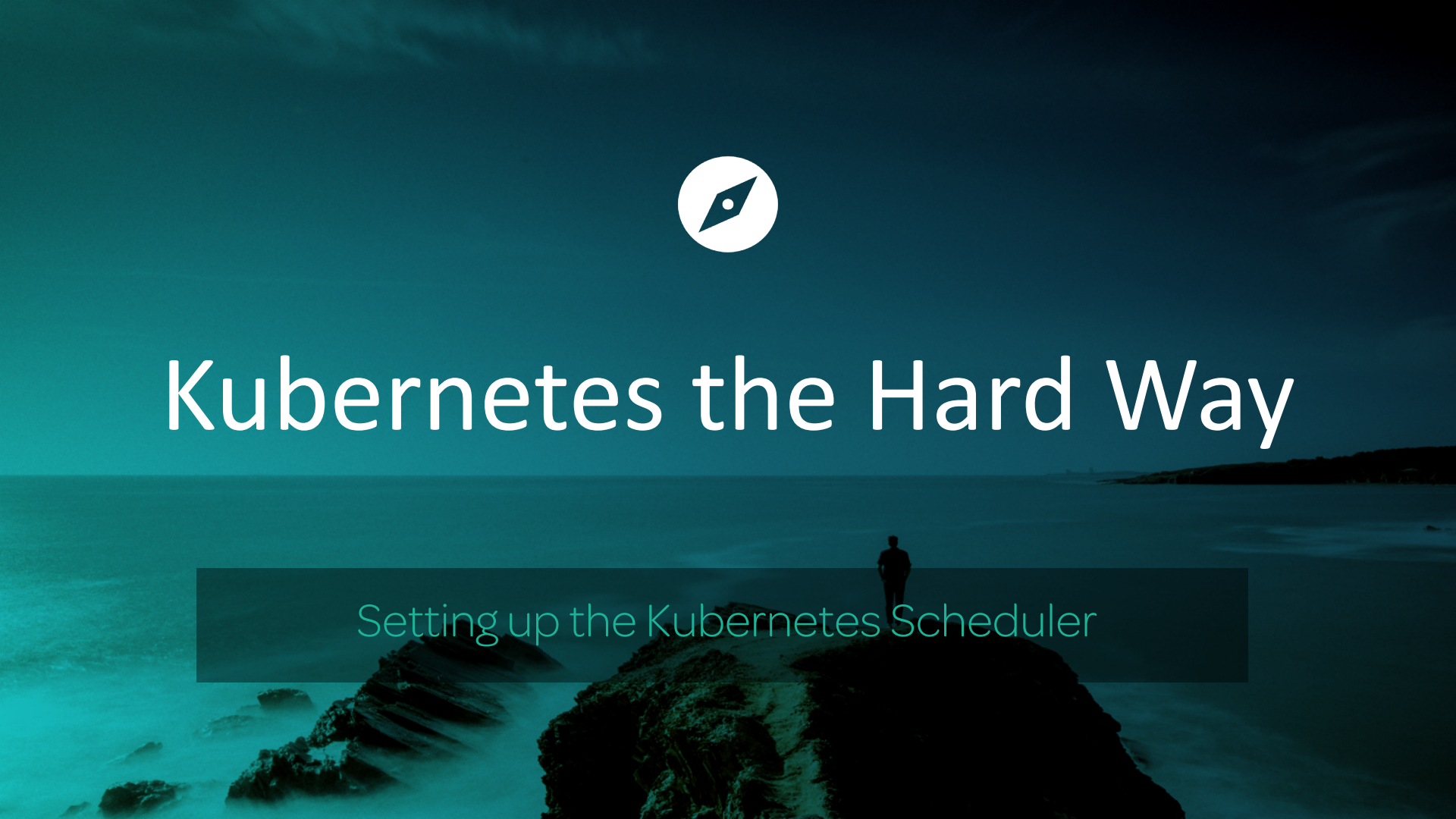
The control plane components will need to be installed on each controller node.

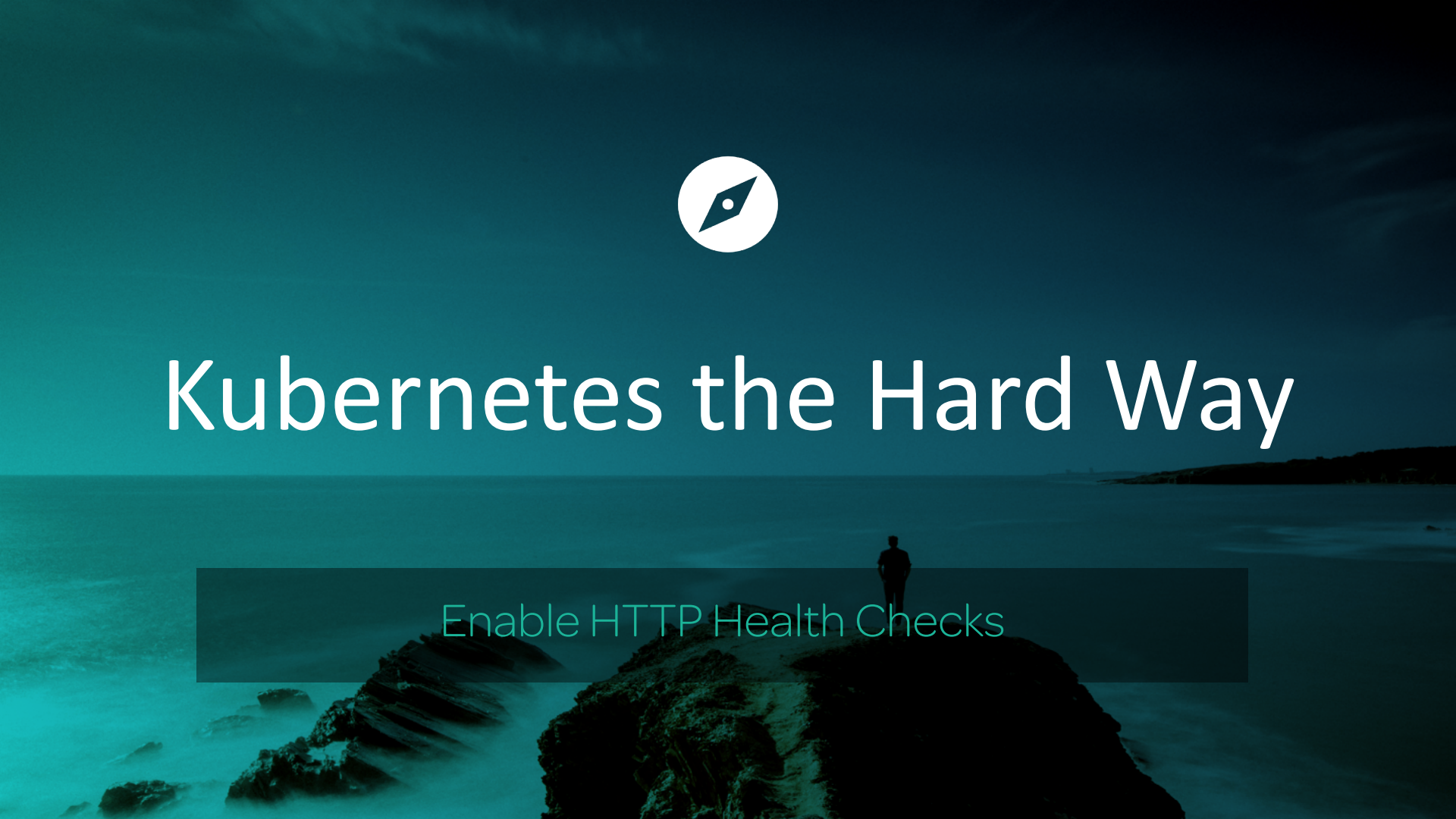

Kubernetes the Hard Way
Control Plane Architecture Overview


Kubernetes the Hard Way
Installing Kubernetes Control Plane Binaries


Kubernetes the Hard Way
Setting up the Kubernetes API Server


Kubernetes the Hard Way
Setting up the Kubernetes Controller Manager


Kubernetes the Hard Way
Setting up the Kubernetes Scheduler

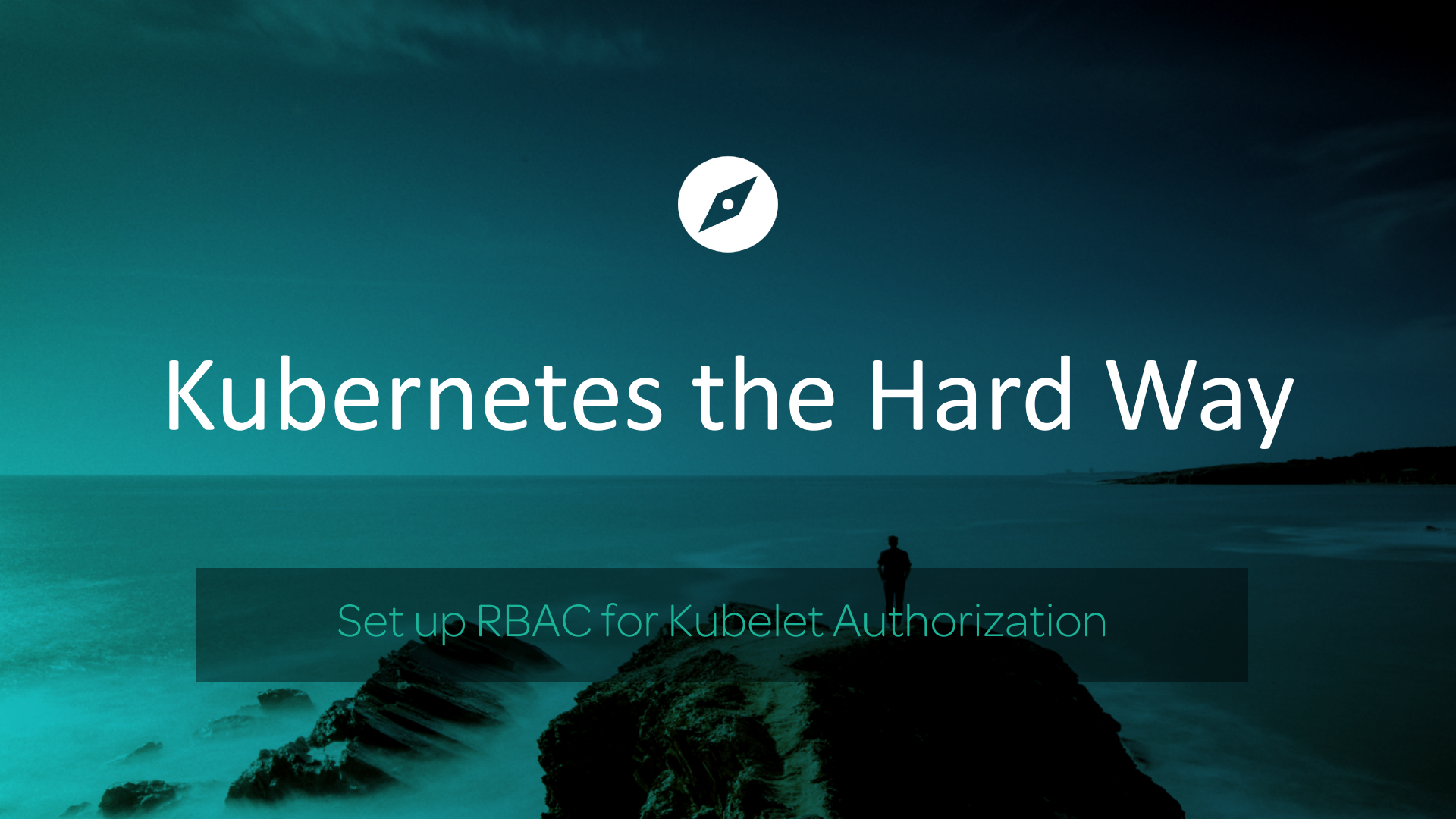

Kubernetes the Hard Way
Enable HTTP Health Checks

Why Do We Need to Enable HTTP Health Checks?

In Kelsey Hightower's original Kubernetes the Hard Way guide, he uses a Good Cloud Platform (GCP) load balancer. The load balancer needs to be able to perform health checks against the Kubernetes API to measure the health status of API nodes.

The GCP load balancer cannot easily perform health checks over HTTPS, so the guide instructs us to set up a proxy server to allow these health checks to be performed over HTTP.

Since we are using Nginx as our load balancer, we don't actually need to do this, but it will be good practice for us. This exercise will help you understand the methods used in the original guide.

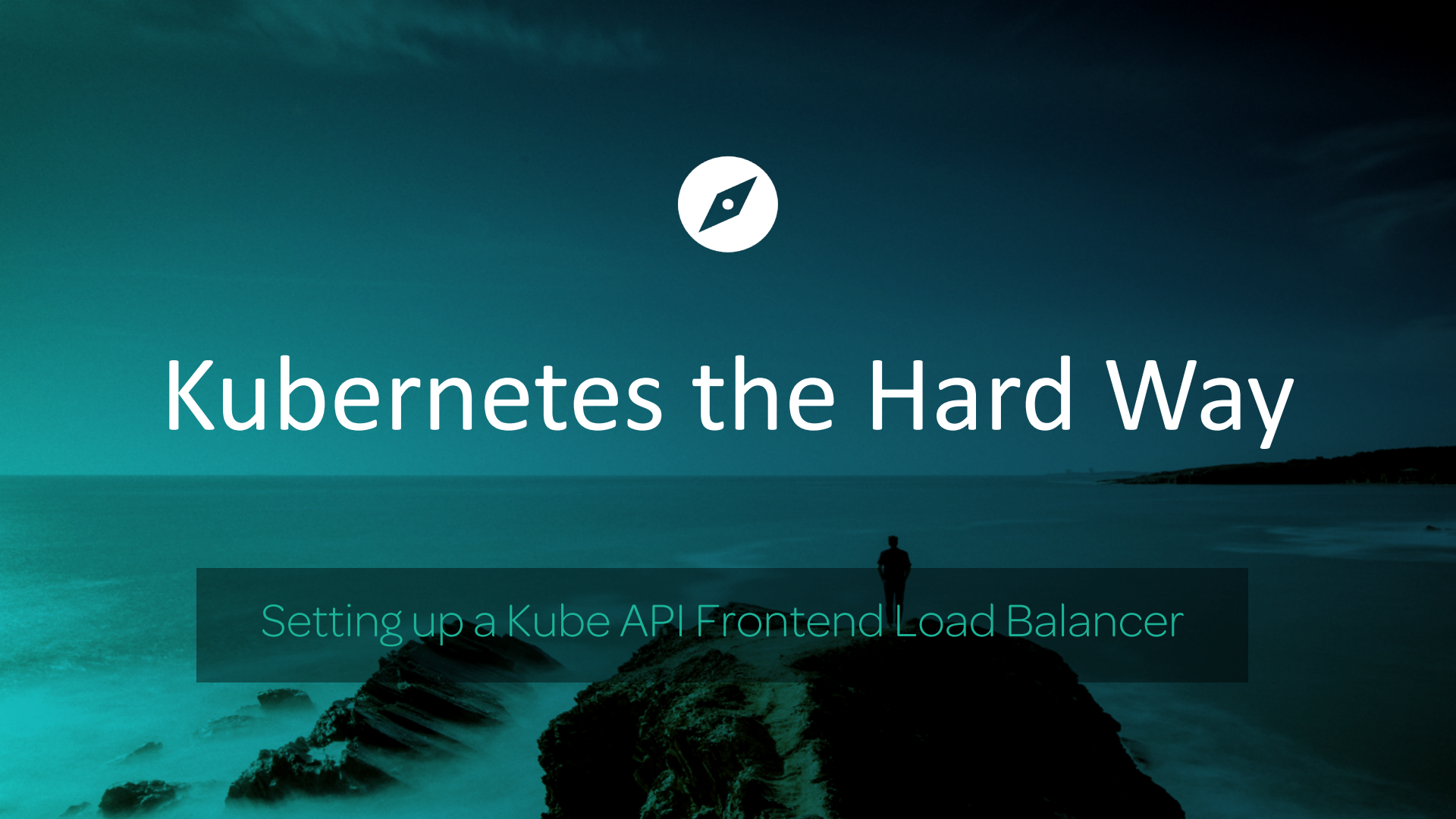

Kubernetes the Hard Way
Set up RBAC for Kubelet Authorization

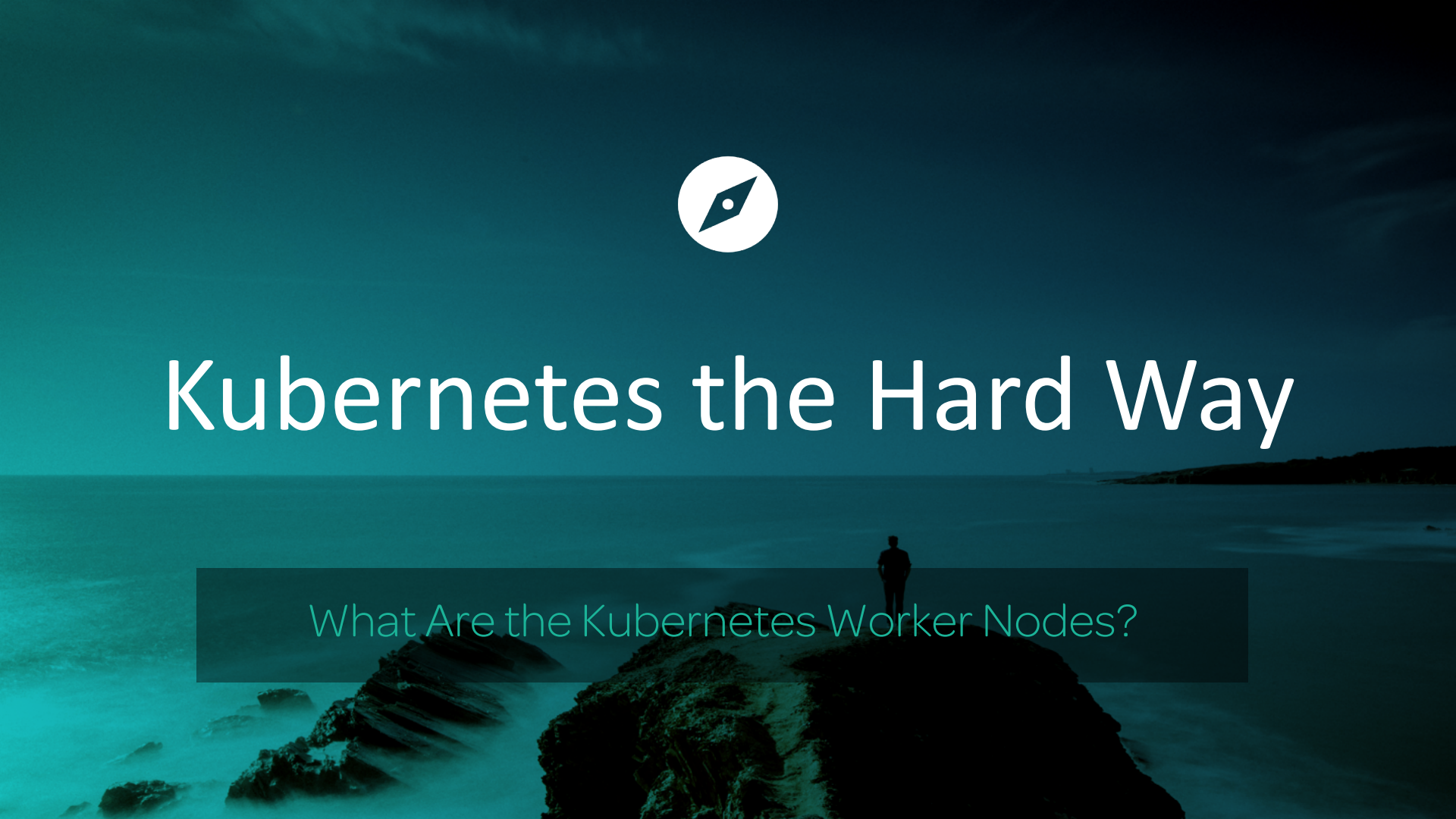
Why Do We Need to Set up RBAC for Kubelet Authorization?

RBAC: Role-Based Access Control

We need to make sure that the Kubernetes API has permission to access the Kubelet API on each node and perform certain common tasks. Without this, some functionality will not work.

We will create a ClusterRole with the necessary permissions and assign that role to the Kubernetes user with a ClusterRoleBinding.


Kubernetes the Hard Way
Setting up a Kube API Frontend Load Balancer



Kubernetes the Hard Way

What Are the Kubernetes Worker Nodes?

What Are the Kubernetes Worker Nodes?

Kubernetes worker nodes are responsible for the actual work of running container applications managed by Kubernetes.

"The Kubernetes node has the services necessary to run application containers and be managed from the master systems."

You can find more information about Kubernetes worker nodes in the Kubernetes documentation:

https://kubernetes.io/docs/concepts/architecture/

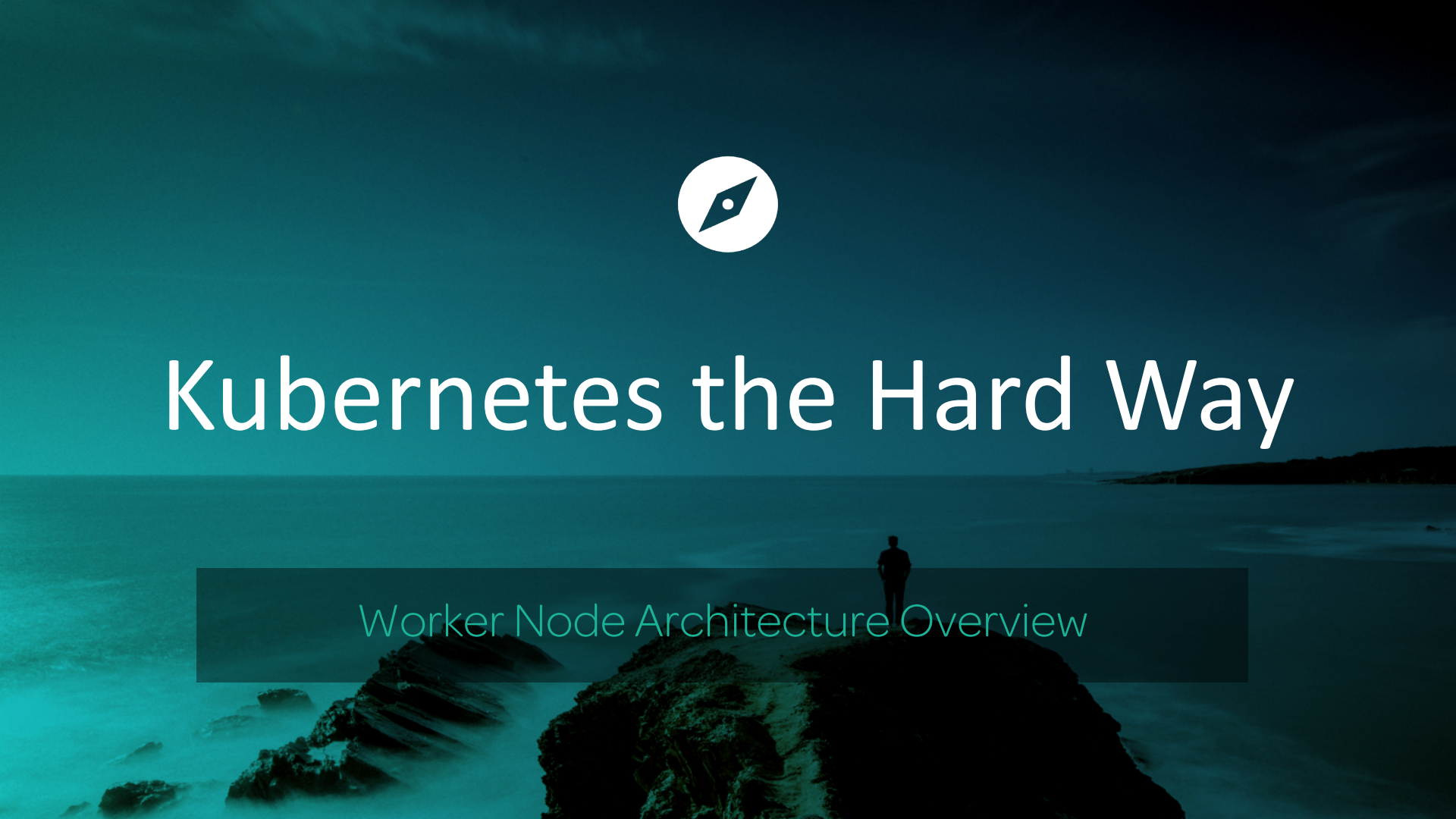
https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md#the-kubernetes-node

Kubernetes Worker Node Components

Each Kubernetes worker node consists of the following components:

- kubelet: Controls each worker node, providing the APIs that are used by the control plane to manage nodes and pods, and interacts with the container runtime to manage containers.
- kube-proxy: Manages iptables rules on the node to provide virtual network access to pods.
- Container runtime: Downloads images and runs containers. Two examples of container runtimes are Docker and containerd (Kubernetes the Hard Way uses containerd).

These components will need to be installed on each worker node.

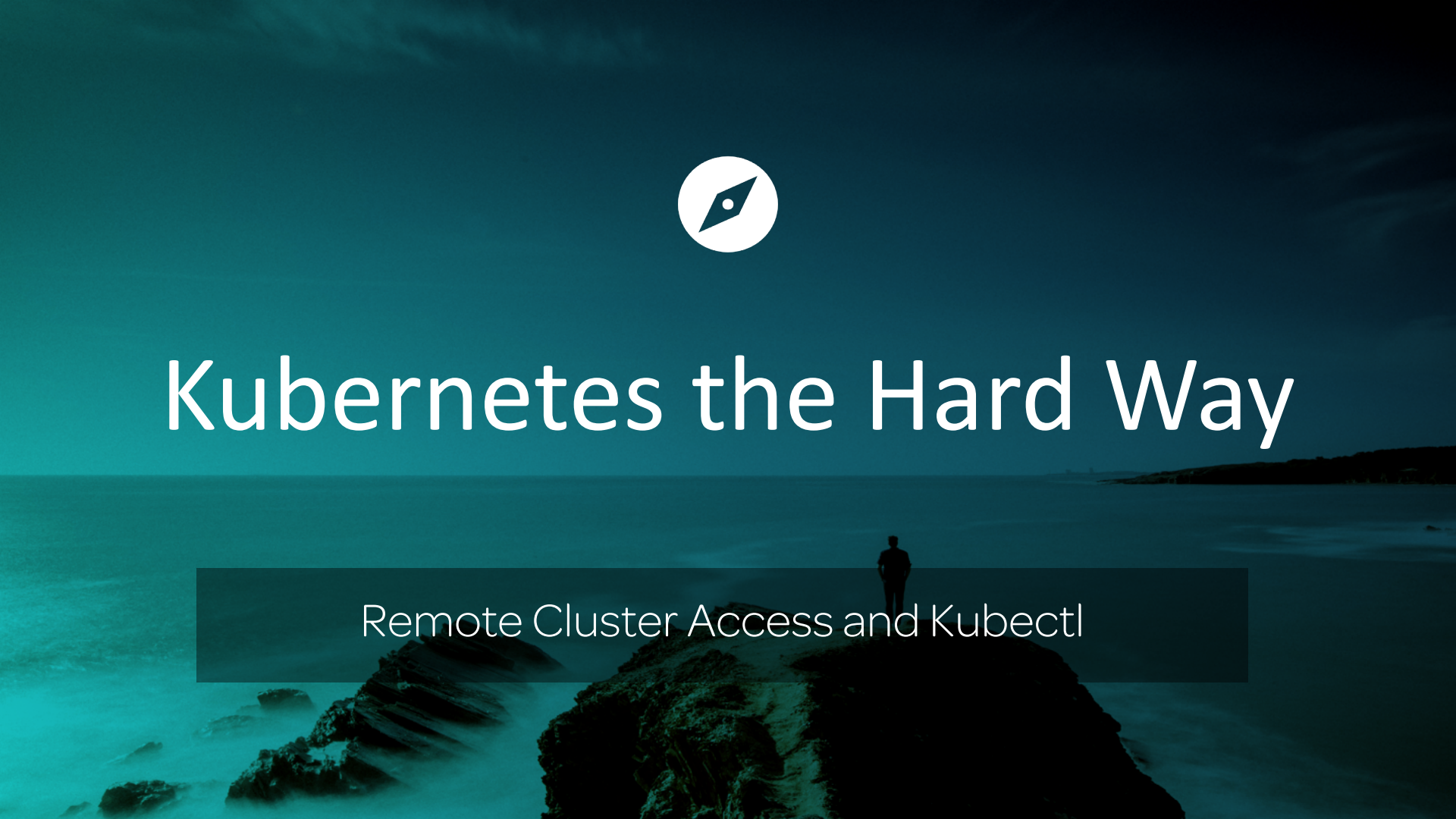

Kubernetes the Hard Way
Worker Node Architecture Overview


Kubernetes the Hard Way
Installing Worker Binaries


Kubernetes the Hard Way
Configuring Containerd


Kubernetes the Hard Way
Configuring Kubelet


Kubernetes the Hard Way
Configuring Kube-Proxy


Kubernetes the Hard Way
Remote Cluster Access and Kubectl

What Is Kubectl?

We have already used kubectl in this course!

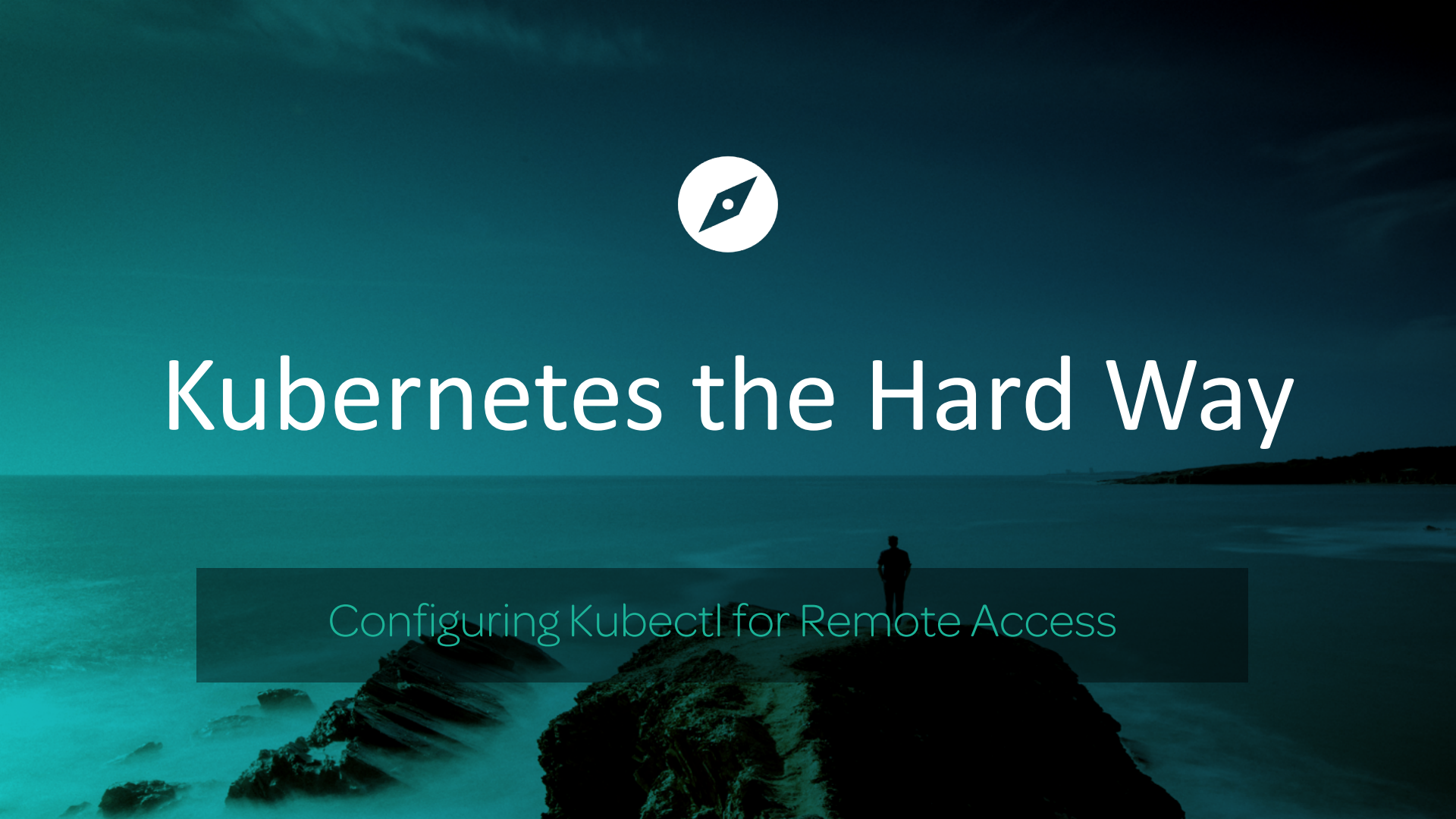
Kubectl is the Kubernetes command line tool. It allows us to interact with Kubernetes clusters from the command line.

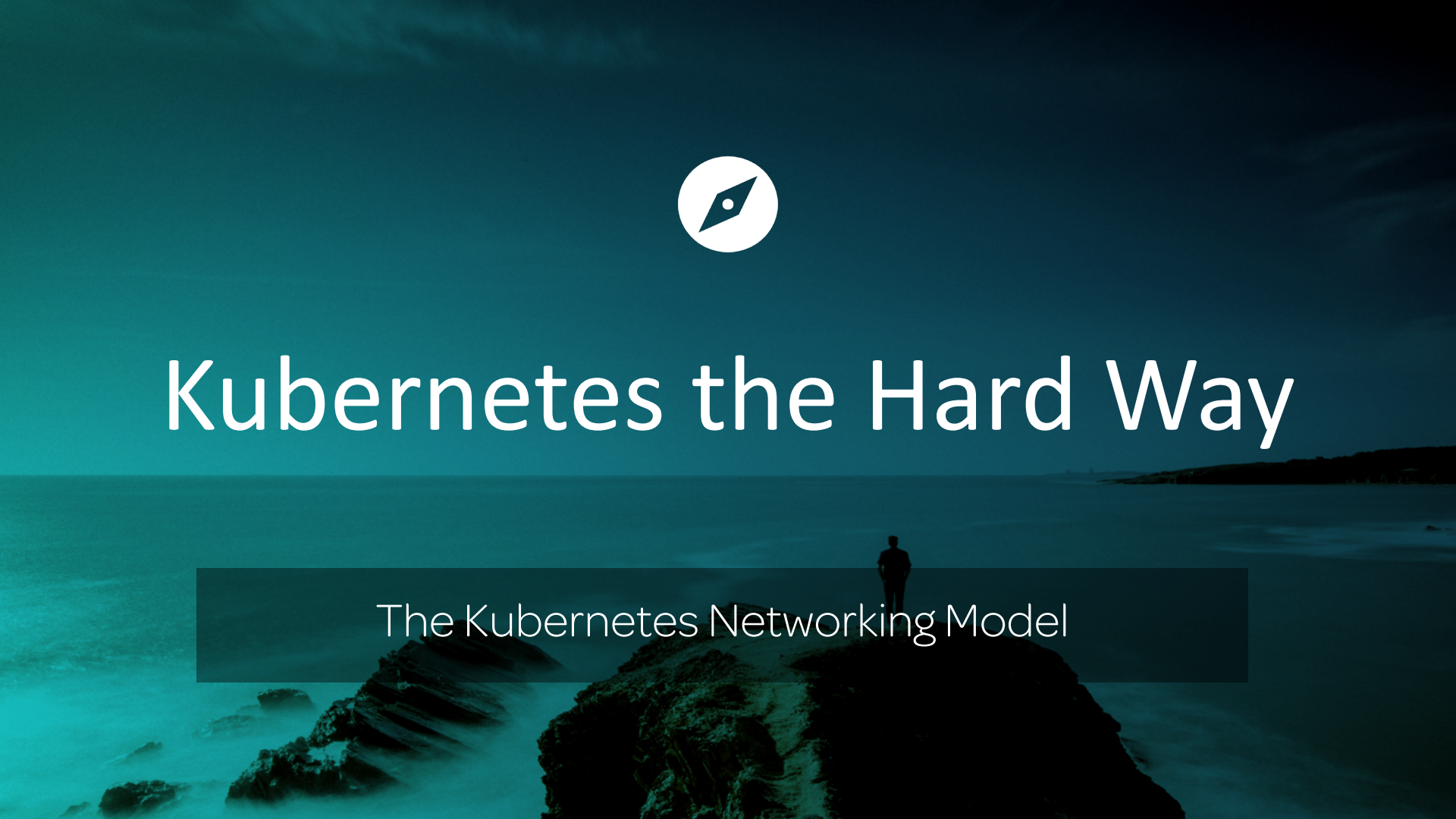
You can find more information about kubectl in the official docs:

https://kubernetes.io/docs/reference/kubectl/overview/

We will set up kubectl to allow remote access from our machine in order to manage the cluster remotely.

To do this, we will generate a local kubeconfig that will authenticate as the `admin` user and access the Kubernetes API through the load balancer.


Kubernetes the Hard Way
Configuring Kubectl for Remote Access



Kubernetes the Hard Way

The Kubernetes Networking Model

The Kubernetes Networking Model

You can find more information on Kubernetes cluster networking in the official docs:

https://kubernetes.io/docs/concepts/cluster-administration/networking/

What Problems Does the Networking Model Solve?

- How will containers communicate with each other?
- What if the containers are on different hosts (worker nodes)?
- How will containers communicate with services?
- How will containers be assigned unique IP addresses? What port(s) will be used?

The Docker Model

Docker allows containers to communicate with one another using a virtual network bridge configured on the host.

Each host has its own virtual network serving all of the containers on that host.

But what about containers on different hosts? We have to proxy traffic from the host to the containers, making sure no two containers use the same port on a host.

The Kubernetes networking model was created in response to the Docker model. It was designed to improve on some of the limitations of the Docker model.

The Kubernetes Networking Model

- One virtual network for the whole cluster.
- Each pod has a unique IP within the cluster.
- Each service has a unique IP that is in a different range than pod IPs.


Kubernetes the Hard Way
Cluster Network Architecture

Cluster Network Architecture

Some Important CIDR ranges:

- Cluster CIDR: IP range used to assign IPs to pods in the cluster. In this course, we'll be using a cluster CIDR of 10.200.0.0/16.
- Service Cluster IP Range: IP range for services in the cluster. This should *not* overlap with the cluster CIDR range! In this course, our service cluster IP range is 10.32.0.0/24.
- Pod CIDR: IP range for pods on a specific worker node. This range should fall within the cluster CIDR but not overlap with the pod CIDR of any other worker node. In this course, our networking plugin will automatically handle IP allocation to nodes, so we do not need to manually set a pod CIDR.

Cluster Network Architecture

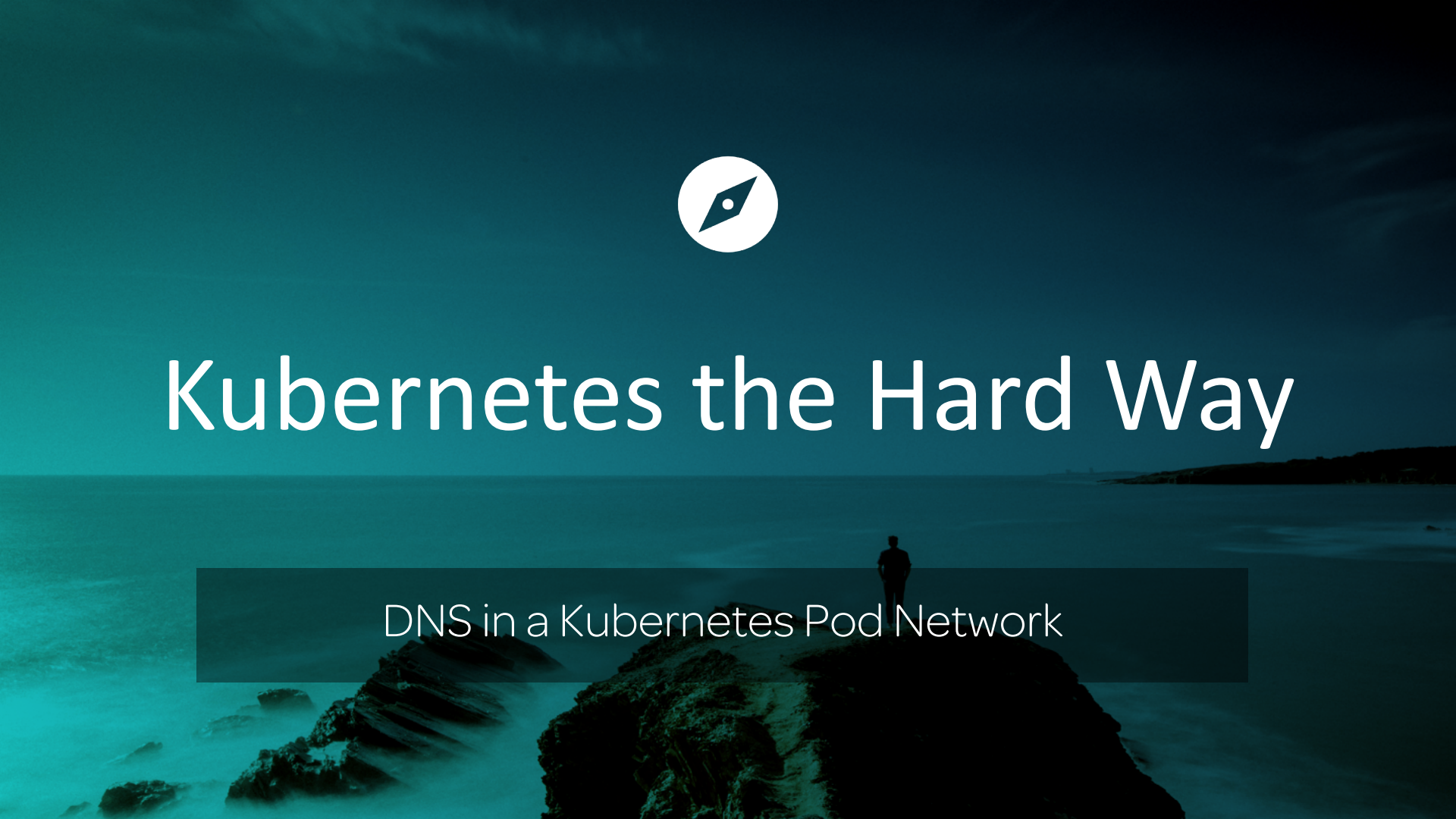
We will be using Weave Net to implement networking in our Kubernetes cluster.

You can find more information about Weave Net here: https://github.com/weaveworks/weave


Kubernetes the Hard Way
Installing Weave Net

Kubernetes the Hard Way

Cleanup


Kubernetes the Hard Way
DNS in a Kubernetes Pod Network

What Does DNS Do Inside a Pod Network?

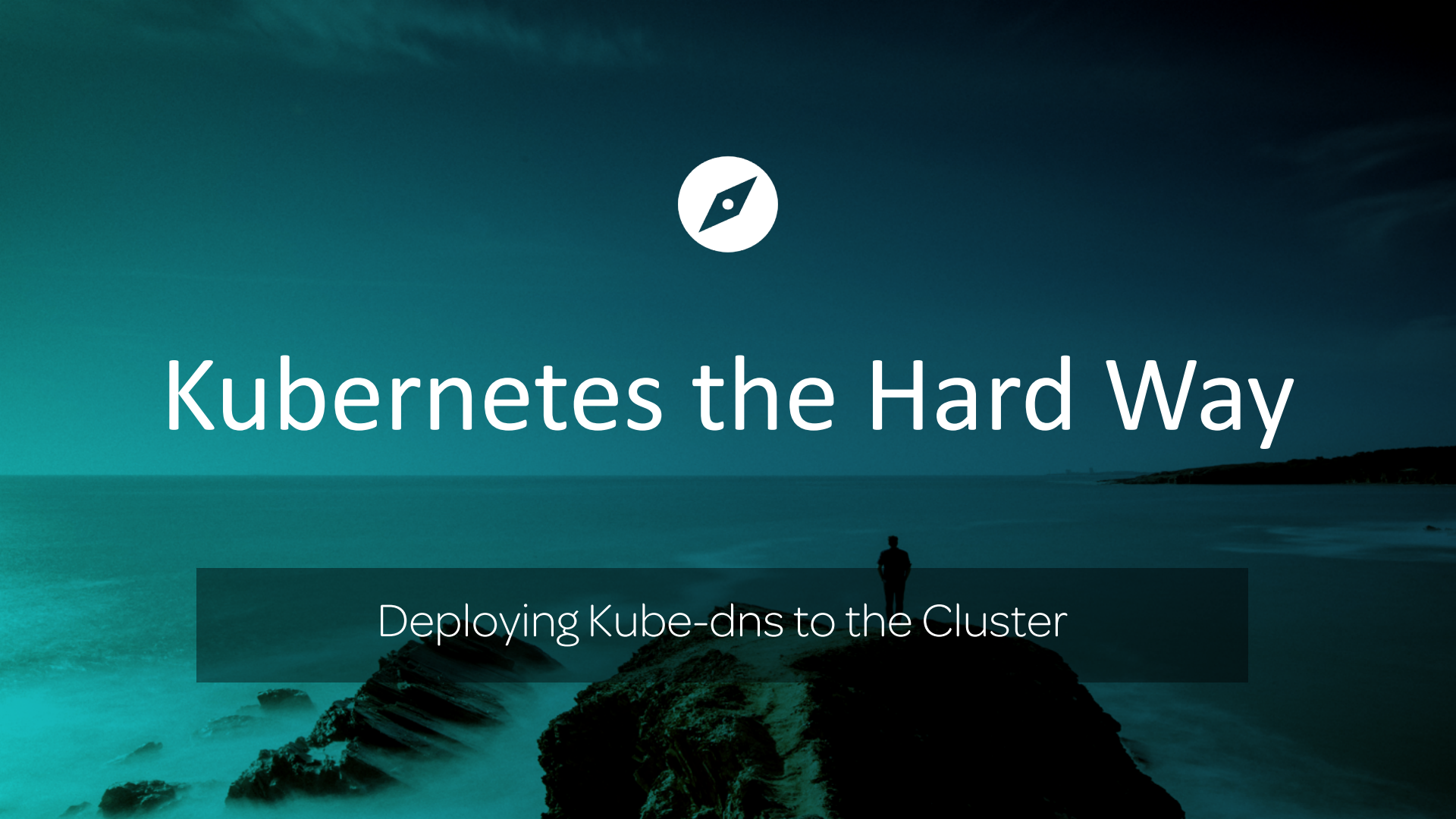
- Provides a DNS service to be used by pods within the network.
- Configures containers to use the DNS service to perform DNS lookups.

For example:

  - You can access services using DNS names assigned to them.
  - You can access other pods using DNS names.

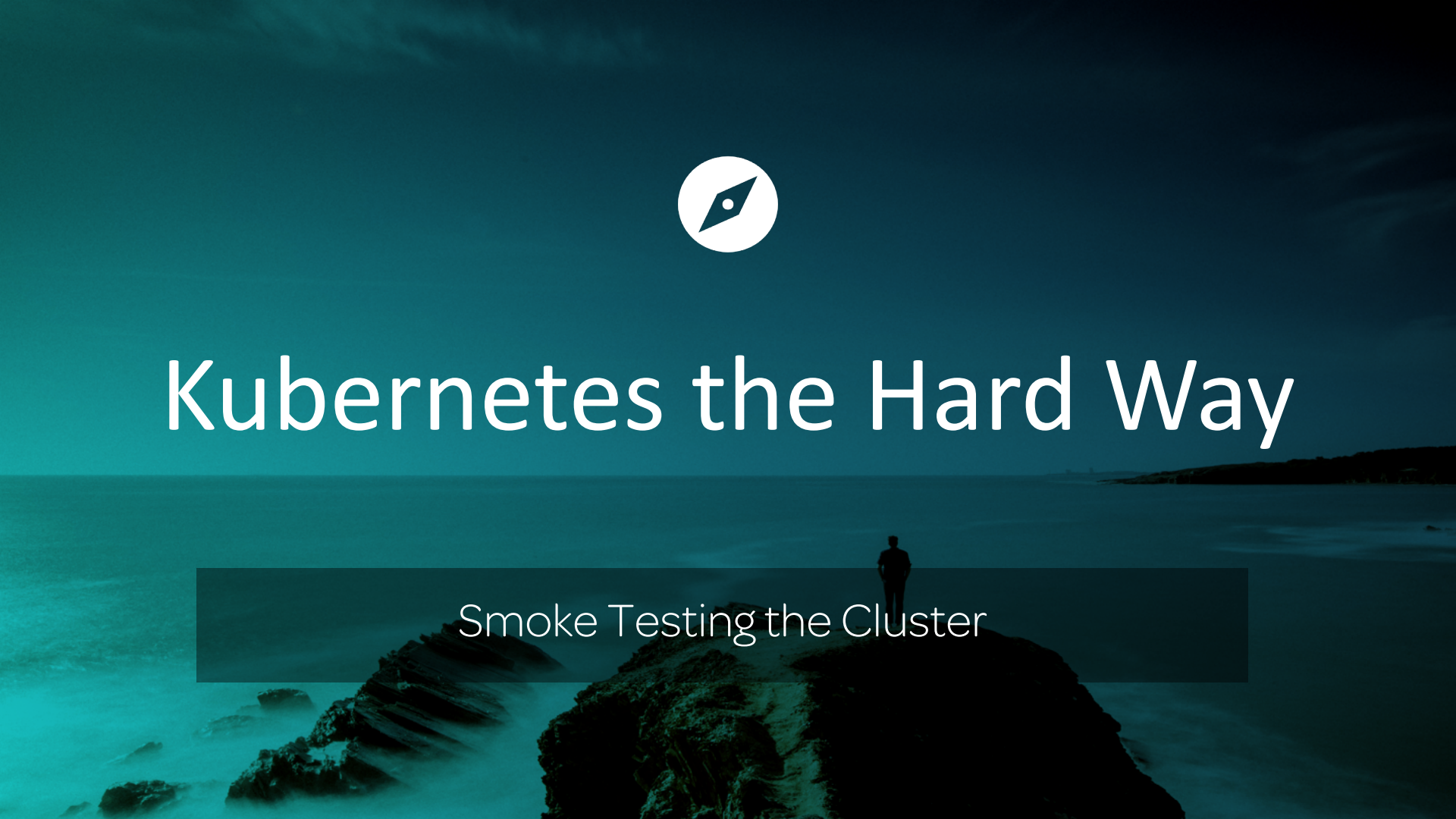
You can find more info in the Kubernetes docs:

https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/



Kubernetes the Hard Way

Deploying Kube-dns to the Cluster



Kubernetes the Hard Way

Smoke Testing the Cluster

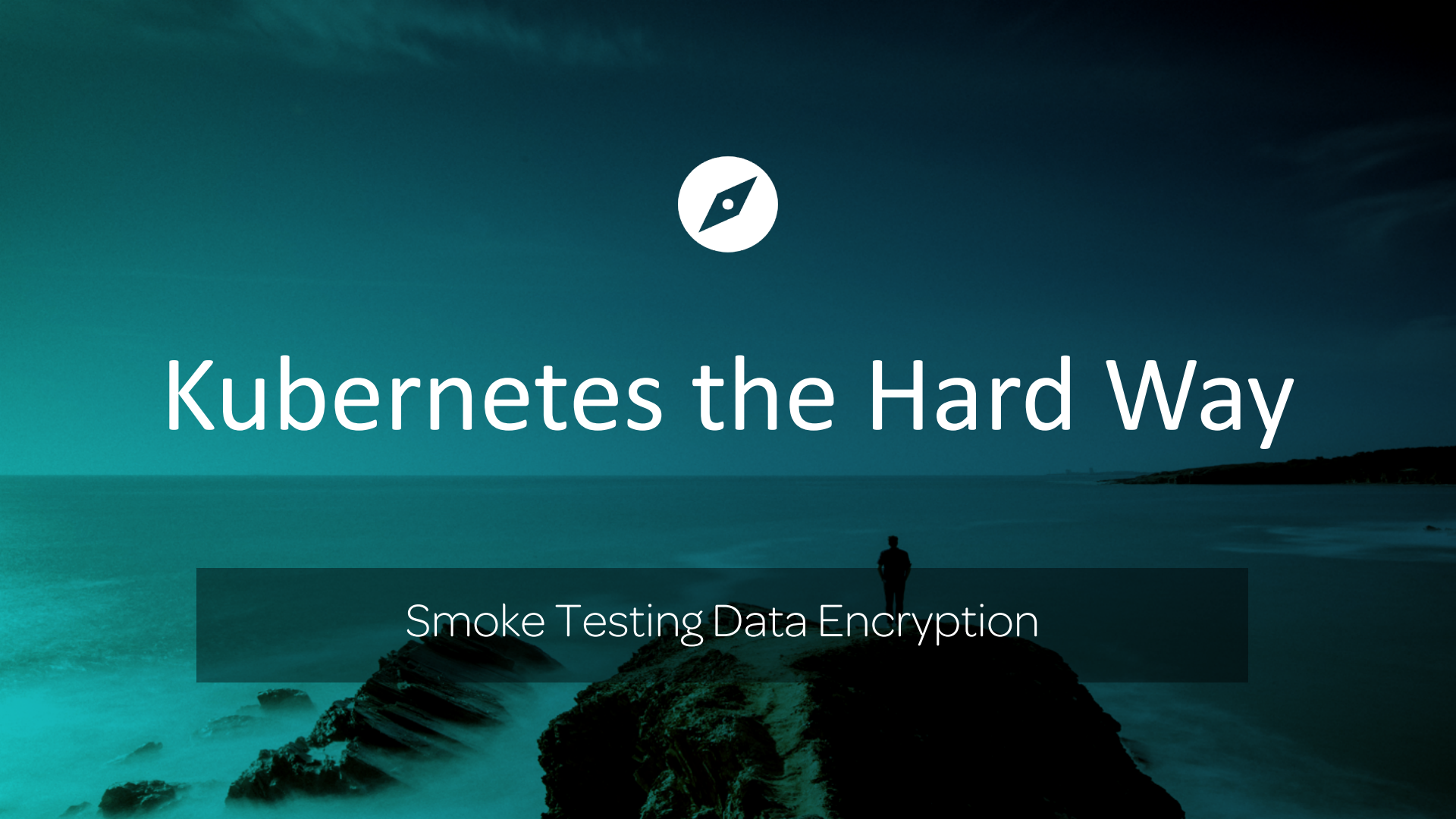
Smoke Testing the Cluster

Congratulations! At this point, we have finished setting up our cluster.

Now we want to run some basic smoke tests to make sure everything in our cluster is working correctly.

We will test the following features:

- Data encryption
- Deployments
- Port forwarding
- Logs
- Exec
- Services
- Untrusted workloads


Kubernetes the Hard Way
Smoke Testing Data Encryption

Smoke Testing Data Encryption

Goal: Verify that we can encrypt secret data at rest.

Strategy:

- Create a generic secret in the cluster.
- Dump the raw data from etcd and verify that it is encrypted.

Kubernetes the Hard Way

Smoke Testing Deployments

Smoke Testing Deployments

Goal: Verify that we can create a deployment and that it can successfully create pods.

Strategy:

- Create a simple deployment.
- Verify that the deployment successfully creates a pod.

Kubernetes the Hard Way

Smoke Testing Port Forwarding

Smoke Testing Port Forwarding

Goal: Verify that we can use port forwarding to access pods remotely.

Strategy:

- Use `kubectl port-forward` to set up port forwarding for an Nginx pod.
- Access the pod remotely with `curl`.


Kubernetes the Hard Way
Smoke Testing Logs

Smoke Testing Logs

Goal: Verify that we can get container logs with `kubectl logs`.

Strategy:

- Get the logs from the Nginx pod container.

Kubernetes the Hard Way

Smoke Testing Exec

Smoke Testing Exec

Goal: Verify that we can run commands in a container with `kubectl exec`.

Strategy:

- Use `kubectl exec` to run a command in the Nginx pod container.


Kubernetes the Hard Way
Smoke Testing Services

Smoke Testing Services

Goal: Verify that we can create and access services.

Strategy:

- Create a NodePort service to expose the Nginx deployment.
- Access the service remotely using the NodePort.


Kubernetes the Hard Way
Smoke Testing Untrusted Workloads

Smoke Testing Untrusted Workloads

Goal: Verify that we can run an untrusted workload under gVisor (runsc).

Strategy:

- Run a pod as an untrusted workload.
- Log in to the worker node that is running the pod and verify that its container is running using runsc.


Kubernetes the Hard Way
Smoke Testing Cleanup


Kubernetes the Hard Way
Kubernetes the Easy Way

Kubernetes the Easy Way

Kubernetes the Hard Way guides you through the process of setting up Kubernetes manually, without any automation or installers. I hope this has taught you a lot about Kubernetes!

In the real world, you probably don't want to do things the hard way. (Work smarter, not harder!) Here are some scripted, automated, or hosted solutions you can use:

- Kubeadm
- Minikube
- Google, AWS, and Azure all provide hosted Kubernetes solutions

You can find more Kubernetes setup solutions here:
https://kubernetes.io/docs/setup/pick-right-solution/

Kubernetes the Hard Way

Next Steps

Next Steps

Now that you have completed Kubernetes the Hard Way, here are some next steps you may want to pursue:

- Get certified with Certified Kubernetes Administrator (CKA)
- Learn more about how Kubernetes can be used in DevOps with Implementing a Full CI/CD Pipeline
- Learn more about Kubernetes and other DevOps tools with LPI DevOps Tools Engineer Certification
- We have lots of new Kubernetes content coming soon, so stay tuned!