Supporting Governance and Security in Enterprise IT through Templating and Infrastructure-as-Code

Christian Frank (#473088)

June 7, 2020

Wirtschaftsinformatik: IT-Management
Dipl. Ing. (FH) Jan Quatram M.A.
FOM - Hochschule für Oekonomie & Management
SS 2020

In this paper, we'll first have a look at Infrastructure-as-Code frameworks. After introducing the most popular tool, Terraform, we'll look at challenges for Enterprises to manage security for container environments. For a possible solution, we'll look at combining Terraform with Cluster Templates provided in Rancher (Rancher Labs) and the integration with CIS benchmarks.

Contents

List of Figures

List of Examples

List of Abbreviations

ARM	Azure Resource Manager
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Deployment
CIS	Center for Internet Security
CLI	Command-Line Interface
CNCF	Cloud Native Computing Foundation
GPU	Graphics Processing Unit
GRC	Governance, risk and compliance
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IT	Information Technology
K3s	Kubernetes on the edge
K8s	Kubernetes
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform as a Service
PSP	Pod Security Policy
RBAC	Role-Based Access Control
REST	Representational State Transfer
RKE	Rancher Kubernetes Engine
SaaS	Software as a Service

SOX	Sarbanes–Oxley Act
YAML	YAML Ain't Markup Language

1 Introduction into Governance and Security

1.1 Pronouns

As we move towards a more gender-fluid society, it's time to rethink the usage of gendered pronouns in scientific texts. Two well-known professors from UCLA, Abigail C. Saguy, and Juliet A. Williams argue that it makes a lot of sense to use singular they/them instead: "The universal singular they is inclusive of people who identify as male, female or nonbinary."[1] Throughout this text, I'll attempt to follow that suggestion and invite my readers to do the same for their papers, and support gender inclusivity through gender-neutral language. A strong focus on diversity and inclusion will significantly benefit your IT organization and help you find and retain talent. Thank you!

1.2 Governance vs. Compliance

In Enterprise IT, it is quite common to use the terms governance and compliance interchangeably. In general, we use the term governance to refer to the process of defining and adhering to a set of operational standards for the overall IT organization. If we look at the formal definition of IT governance in ISO/IEC 38500:2015, though, IT governance is much more focused on the business aspect of running IT. As defined by ISO/IEC, IT governance will have a heavy emphasis on budgetary control, financial performance, and investments. Adherence to standards, such as the Sarbanes–Oxley Act, is more seen as part of compliance than of governance.

However, governance and compliance are part of the overall governance, risk, and compliance (GRC) practice in an IT organization and have to go hand in hand. Governance will include compliance, and compliance needs governance (oversight).

Throughout the remainder of the document, I will use the term governance to refer to all three aspects, governance, risk, and compliance.

1.3 Security

In the last decade, IT operations have seen their primary focus shift from an on-premise, dedicated environment to a shared, on-demand infrastructure, to cloud computing, provided by public cloud providers.

The National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared

[1] *Saguy, A. (2020)*: Why We Should All Use They/Them Pronouns. [12]

pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[2]

In the public cloud space, three major players have emerged: Amazon Web Services, Microsoft Azure, and Google Cloud. A big contender is Alibaba Cloud who, like Huawei, struggle with limited acceptance because of their (perceived) relationship to the Chinese government.

In the early days of cloud computing, virtualization played a key role and drove adoption; a couple of years ago, the technology focus shifted away from virtualization towards containerization and the Kubernetes orchestration framework. All major Hyperscalers now have container run-time offerings, managed and non-managed, based on Kubernetes. Google has a clear advantage since they are the original creators of Kubernetes - Kubernetes is a spin-off of an internal orchestration tool, Borg, that Google uses to run google.com.

Container run-time environments have important security aspects, as outlined by NIST in their container security guide.[3]

For this document, we'll concentrate on the aspects of security and governance (regulatory compliance) in container run-time environments, focusing on Kubernetes as it is the dominant choice at the time of writing.

Furthermore, we'll concentrate on the run-time layer itself and not cover hardening aspects of the underlying computing infrastructure. Not that this is not an important topic in itself; it would just exceed the scope of this paper.

[2] *Mell, P. (2011)*: The NIST Definition of Cloud Computing. [5]
[3] See *Souppaya, M. (2017)*: Application Container Security Guide. [14]

2 The need for Governance and Security in container run-time environments

2.1 Container Run-Time Environments

Let's start with the basics. In most of IT, Kubernetes[4] has emerged as the container run-time environment of choice.

Earlier competitors, such as Docker Swarm or Mesos DC/OS, have all but vanished. After Mirantis acquired Swarm from Docker last year, they announced a two-year end-of-life timeline for support, but they have since announced that they might extend this and possibly invest in new features.

The original platforms that spawned cloud-native computing and application design and gave us the concept of micro-services, Heroku and Cloud Foundry, have also slightly gone out of favor, or, in the case of Cloud Foundry, moved to a Kubernetes-based run-time. Salesforce.com acquired Heroku in 2010; an independent foundation, led by VMware, owns Cloud Foundry with SUSE playing a very active part.

In a previous paper, I have covered Kubernetes in more detail[5], in this paper I will only cover the necessary basics.

2.2 Kubernetes Architecture

A Kubernetes container run-time environment, or cluster, in general, will consist of one or more logically grouped systems. These systems, most likely virtual machines, will run a supported container run-time, such as Docker (Microsoft / Ubuntu) or Podman (Red Hat).

A single Kubernetes cluster has a Control Plane and an Execution Plane. Usually, there are three nodes in the control plane and one or more nodes in the execution plane. For the control plane, you should choose an odd number of nodes because the underlying etcd database is a distributed system and needs to reach quorum on startup. The number of worker nodes in the execution plane only depends on the needs of your application.

The Kubernetes control plane takes care of orchestrating the application deployments and maintains their state; on the other hand, the worker nodes execute the actual application containers as defined and scheduled by the control plane. Worker nodes can

[4]See *The Linux Foundation (2020)*: Production-Grade Container Orchestration. [17]
[5]See *Frank, C. (2020)*: Multi-Cluster Management fuer Containerumgebungen .[2]

be organized into one or more node pools to separate hardware features or reliability aspects.

In addition to providing compute nodes, Kubernetes includes overlay networks to allow communication between the cluster nodes, and ingress controllers for inbound network access to the running applications. Persistent storage, even though it's an anathema to stateless, cloud-native computing, is provided through persistent volumes and storage classes and linked to the underlying cloud provider.

All administrative access to a given Kubernetes cluster is via the master nodes and its Kubernetes API endpoint.

2.3 Kubernetes Security

Installing the first Kubernetes cluster is no longer a big task, especially on the three big public cloud providers, which all offer managed Kubernetes clusters with easy, one-click installations.

However, designing the platform landscape requires some architectural knowledge and should be planned well in advance.

One of the crucial components of security when running containers in production, according to NIST, is the separation between applications and systems. Put careful consideration during design into the number and type of Kubernetes clusters deployed.[6]

Kubernetes itself, at the time of writing, does not provide for hard tenancy. Breakouts and cross-talk remain a possibility. To entirely separate applications on all layers (compute, network, and storage), using multiple clusters is a good option. Many enterprises might thus end up with more than one Kubernetes cluster, sometimes with many more. This will, of course, have a significant effect on operations.

With the increase in popularity, Kubernetes also brings other new security challenges that should be considered when designing the IT environment.[7]

We're observing a trend in Enterprise IT to move from traditional perimeter-based network security to more modern zero- or low-trust networks with identity-based security and temporary infrastructure. Having multiple, short-lived Kubernetes cluster is thus an entirely likely scenario going forward.

[6]See *Weibel, D. (2020)*: Architecting Kubernetes clusters. [18]
[7]See *Weizmann, Y. (2020)*: Threat matrix for Kubernetes. [19]

2.4 Rancher Overview

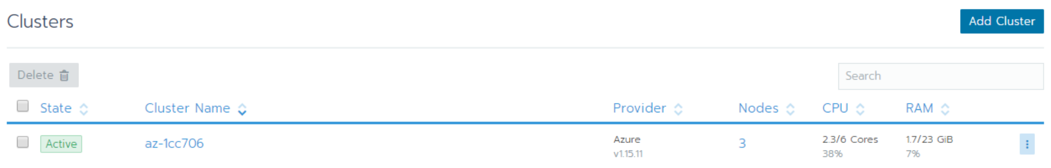
To easily manage a set of Kubernetes clusters, we can turn to Rancher.

What is Rancher? According to the Rancher Labs website, it is "[...] a complete software stack for teams adopting containers. It addresses the operational and security challenges of managing multiple Kubernetes clusters, while providing DevOps teams with integrated tools for running containerized workloads"[8]

Rancher provides a management platform to centrally manage multiple Kubernetes clusters in Enterprise IT, all from a choice of two user-friendly GUIs. Rancher also offers integration tools for application development and robust enterprise-grade features for security and governance. For operations, Rancher provides integrated solutions for logging, monitoring, and auditing, together with many other features, such as CIS scans or a built-in service mesh.

The classic Rancher GUI looks like this:

Figure 1: Rancher Overview



State	Cluster Name	Provider	Nodes	CPU	RAM
Active	az-1cc706	Azure v1.15.11	3	2.3/6 Cores 38%	1.7/23 GiB 7%

2.5 Rancher Architecture

Rancher is, similar to Kubernetes, itself a containerized application and can be installed from a single image to a single Docker host. Such a single-node installation is ideal for test-beds or local Rancher installations on a laptop, for example. A single node installation does not provide any redundancy in case of failure.

For production installations, Rancher can be installed on a separate Kubernetes cluster, using Kubernetes' redundancy mechanisms for high-availability and resiliency. It could either be co-hosted on an existing cluster or better, on a small, separate infrastructure cluster. Installation on a separate cluster is the recommended choice. It is good prac-

[8] *Rancher Labs (2019)*: Run Kubernetes Everywhere. [11]

tice in IT to keep administrative tools on separate infrastructure from the administered infrastructure, and thus the installation on a different cluster is the most widespread.

In addition to the GUI, the Rancher server also provides a central Kubernetes API endpoint, which prevents unauthorized access and sits as an intermediary between the users and the actual Kubernetes clusters.

You can find more details on the Rancher architecture in Rancher's technical architecture document.[9]

The open-source Kubernetes run-time threat detection engine Falco works very well with Rancher.[10]

I've covered the aspect of cluster and workload segmentation in more detail in my previous FOM paper on multi-cluster management in Enterprise IT. I've also explained in that paper as to why I view the combination of Kubernetes and Rancher as a very beneficial one, especially for Enterprise IT.[11]

[9]See *Rancher Labs (2020)*: Rancher Technical Architecture. [10]
[10]See *Shankar, P. (2020)*: Runtime Security in Rancher with Falco. [13]
[11]See *Frank, C. (2020)*: Multi-Cluster Management fuer Containerumgebungen .[2]

3 Infrastructure as Code

3.1 Terraform

All major cloud providers have their infrastructure scripting tools (Azure Resource Manager, AWS Cloud Formation). Still, there's one declarative tool that's available for all infrastructure platforms, on-premise or public: Terraform by HashiCorp.[12].

As of Rancher 2.3, released in October 2012, Rancher has a stable Terraform provider.[13] You can easily create and decommission Kubernetes clusters from a Terraform plan as part of a move of Enterprise IT to Infrastructure-as-Code.

With Terraform, creating infrastructure becomes as easy as writing a piece of code. Terraform integrates nicely with your existing source code revision systems, such as GitHub or GitLab. It will also ensure that all infrastructure deployments are repeatable and uniformly executed.

Adding Terraform to the container run-time tool chest of Rancher and Kubernetes is a major benefit and huge step forward, as it will make the creation of new Kubernetes clusters so much easier.[14]

3.2 Templates

Another essential piece for your Kubernetes tool chest is templates. Templates will allow you to define certain configuration items across the organization, which will help a lot when dealing with hardening in the next chapter. By using templates and establishing policies that make their use mandatory, you can enforce settings and hide some details, such as credentials.

Rancher offers templates on several levels:

- Cloud Credentials
- Node Templates
- Cluster Templates

We'll look at each of these in more detail below, but let's first look at the different possibilities of creating a Kubernetes cluster from within Rancher.

[12]See *HashiCorp (2019)*: Deliver infrastructure as code with Terraform. [3]
[13]See *Rancher Labs (2019)*: Introducing the Rancher 2 Terraform Provider. [8]
[14]See *Frank, C. (2020)*: Deploy Kubernetes Clusters on Microsoft Azure with Rancher. [1]

3.3 Kubernetes Cluster Creation

3.3.1 Managed Kubernetes

Overall there are three options to create a Kubernetes cluster with Rancher on any of the big public cloud providers:

- Managed Kubernetes (GKE, AKS, EKS)
- Rancher node-driver (Azure, AWS)
- Custom nodes (GCP, Azure, AWS)

To create a managed Kubernetes cluster, you'll have to follow the following steps:

First, within a Terraform plan, define the Rancher provider. Then, in the Rancher cluster definition, define the GKE, AKS, or EKS options and finally have Terraform create the cluster. Terraform will be using the platform API for this.

In this setup, the cloud provider will manage the Kubernetes control plane, and Rancher's functionality will be a bit more limited than in the next option.

3.3.2 Rancher Node-Driver

If you're happy to have Rancher manage the control plane and have full control, and are on either AWS or Azure, consider using the Rancher node-driver. To do this, following the following steps:

First, within a Terraform plan, define the Rancher provider. Second, in the Rancher cluster definition, define Azure or AWS cloud provider within the RKE cluster options. Then, create the appropriate cloud credentials and node templates. Finally, have Terraform create the cluster, using docker-machine.

Using the Rancher node-driver is the most preferred option and the one that we will follow throughout the document.

3.3.3 Custom Nodes

If you need more fine-grained control over the underlying infrastructure, Rancher also offers the ability to use custom created nodes for its Kubernetes clusters. For this, you'll need to follow these steps:

First, within a Terraform plan, define both the Rancher provider and an infrastructure provider. Second, in the Rancher cluster definition, define the cloud provider within the

RKE cluster options. Then, have Terraform create the cluster nodes with the infrastructure provider and pass the Rancher registration command to cloud-init

Although this is the most flexible approach, Rancher will lose the ability to scale the node pools, among other features. It also requires that the Terraform plans have access to credentials for the infrastructure provider, unlike in the option above, where you could provide them centrally in Rancher.

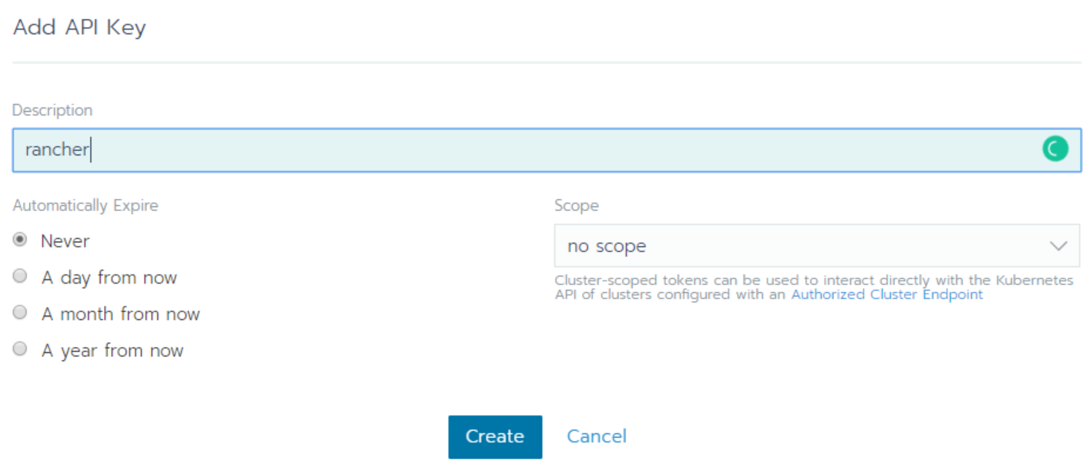
3.3.4 Import

A fourth and final option would be to create clusters completely outside of Terraform and Rancher and manually import them into Rancher later. Importing clusters is an entirely valid option, but outside of our document's scope, it would require other forms of automation.

3.4 Rancher Provider

Let's start defining our Kubernetes cluster. The first step is to set up the credentials for Rancher. All API access to Rancher is controlled from the built-in RBAC controller, and every user can create their own API keys.

Figure 2: Rancher API Key



Add API Key

Description: rancher

Automatically Expire: [✓ Never] A day from now A month from now A year from now

Scope: no scope

Cluster-scoped tokens can be used to interact directly with the Kubernetes API of clusters configured with an Authorized Cluster Endpoint

We'll then take the defined token and create the Rancher provider in our plan file `provider.tf`:

Example 1: Rancher Provider

```
# Rancher
provider "rancher2" {
  api_url = var.rancher-url
  token_key = var.rancher-token
}
```

The code above is all the end-user credential and provider setup we'll need to create a Kubernetes cluster. On `terraform init` the Rancher provider library will be downloaded and initialized.

3.5 Cloud Credentials

It is good practice to keep the provider definitions separate from the main plan, so from here on, all example plan code will go into the main plan file, `main.tf`. Also, from here on, the example plan code will be specific to deployment on Microsoft Azure, but you can easily adapt it for Amazon Web Services.

The first template we want to define is the cloud credentials. Regardless of the cloud provider, we need to provide some form of access credentials. On Microsoft Azure, this has to be done in the form of a service principal. These credentials could be created by the user, or better, be pre-provisioned by the IT organization.

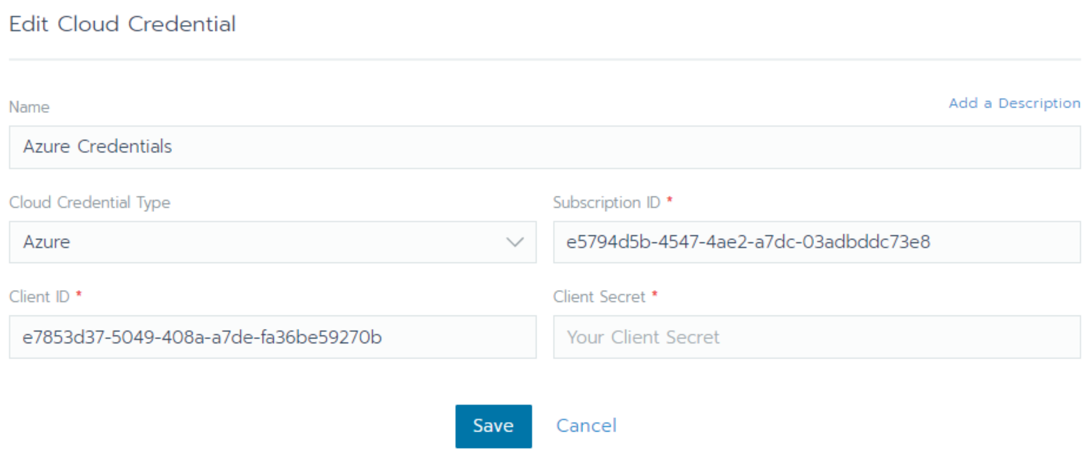
In our Terraform plan, creating the credentials looks like this:

Example 2: Cloud Credentials

```
# Rancher cloud credentials
resource "rancher2_cloud_credential" "credential_az" {
  name = "Azure Credentials"
  azure_credential_config {
    client_id = var.az-client-id
    client_secret = var.az-client-secret
    subscription_id = var.az-subscription-id
  }
}
```

To create credentials we can also use the Rancher GUI with the same input fields:

Figure 3: Cloud Credentials


Edit Cloud Credential

Name: Azure Credentials

Cloud Credential Type: Azure

Subscription ID *: e5794d5b-4547-4ae2-a7dc-03adbddc73e8

Client ID *: e7853d37-5049-408a-a7de-fa36be59270b

Client Secret *: Your Client Secret

3.6 Node Templates

As we've learned earlier, a Kubernetes cluster consists of one or more node pools, at least one for the control plane and one or more for the worker nodes. On a small installation, the control plane and the workers can be in the same node pool.

To create a node pool, we first need to define node templates in our plan:

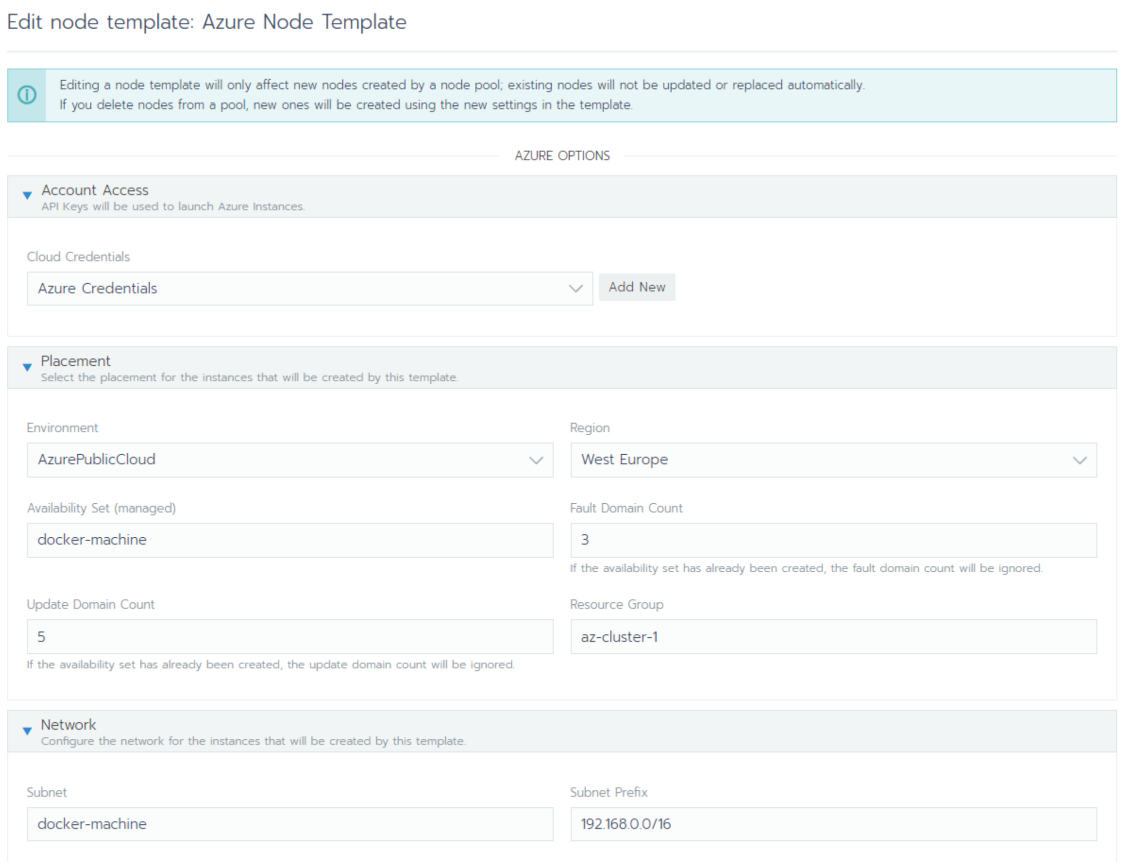
Example 3: Node Template

```
# Rancher node template
resource "rancher2_node_template" "template_az" {
  name = "Azure Node Template"
  cloud_credential_id = rancher2_cloud_credential...id
  engine_install_url = var.dockerurl
  azure_config {
    disk_size = var.disksize
    image = var.image
    location = var.az-region
    managed_disks = true
    open_port = var.az-portlist
    resource_group = var.az-resource-group
    storage_type = var.az-storage-type
    size = var.type
  }
}
```

In a node template, we define size, OS image, and regional placement of the node. Depending on the future role, we can define them as small or as big as we need them, or define nodes with specialized hardware, such as GPUs, for machine-learning tasks.

We can make the same definition of node pools as above with the Rancher GUI:

Figure 4: Node Template



Edit node template: Azure Node Template

Editing a node template will only affect new nodes created by a node pool; existing nodes will not be updated or replaced automatically.
If you delete nodes from a pool, new ones will be created using the new settings in the template.

AZURE OPTIONS

Account Access
API Keys will be used to launch Azure Instances.

Cloud Credentials: Azure Credentials

Placement
Select the placement for the instances that will be created by this template.

Environment: AzurePublicCloud
Region: West Europe
Availability Set (managed): docker-machine
Fault Domain Count: 3
If the availability set has already been created, the fault domain count will be ignored.
Update Domain Count: 5
If the availability set has already been created, the update domain count will be ignored.
Resource Group: az-cluster-1

Network
Configure the network for the instances that will be created by this template.

Subnet: docker-machine
Subnet Prefix: 192.168.0.0/16

The actual node pools will be created later; the template only defines their physical characteristics.

3.7 Cluster Templates

The third and final template that we are going to define is the template for the actual cluster.

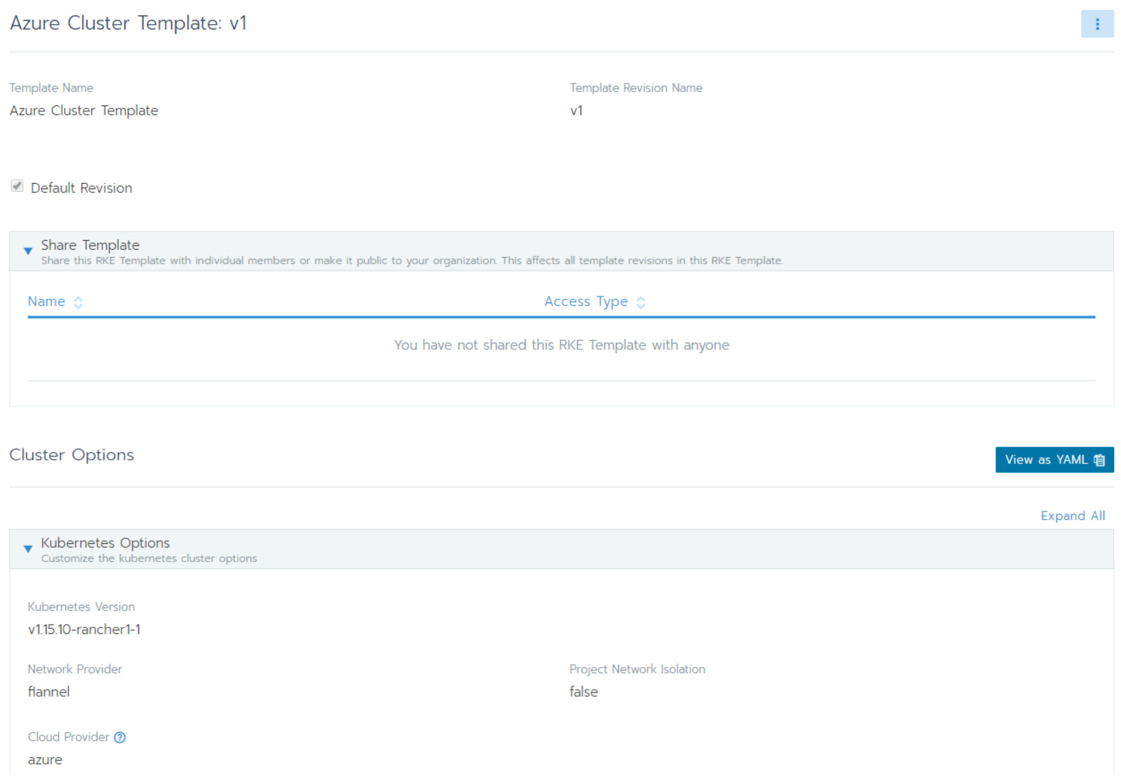
Example 4: Cluster Template

```
# Rancher cluster template
resource "rancher2_cluster_template" "template_az" {
  name = "Azure Cluster Template"
  template_revisions {
    name = "v1"
    default = true
    cluster_config {
      cluster_auth_endpoint {
        enabled = false
      }
      rke_config {
        kubernetes_version = var.k8version
        ignore_docker_version = false
        cloud_provider {
          name = "azure"
          azure_cloud_provider {
            aad_client_id = var.az-client-id
            aad_client_secret = var.az-client-secret
            subscription_id = var.az-subscription-id
            tenant_id = var.az-tenant-id
            resource_group = var.az-resource-group
          }
        }
      }
    }
  }
}
```

In this template, we define all the essential characteristics of our Kubernetes clusters that we want to enforce across the IT organization. For example, in the plan above, we do not allow users to bypass Rancher and access the Kubernetes clusters directly. This setting will give us a solid first line of defense against malicious attacks based on access credentials.

If you prefer the Rancher GUI, you can create cluster templates there, too:

Figure 5: Cluster Template


Azure Cluster Template: v1

Template Name
Azure Cluster Template

Template Revision Name
v1

✓ Default Revision

Share Template
Share this RKE Template with individual members or make it public to your organization. This affects all template revisions in this RKE Template.

Name	Access Type
You have not shared this RKE Template with anyone	

Cluster Options

Kubernetes Options
Customize the kubernetes cluster options

Kubernetes Version
v1.15.10-rancher1-1

Network Provider
flannel

Project Network Isolation
false

Cloud Provider
azure

3.8 Kubernetes Cluster

Now we have all the components together to build a Kubernetes cluster - credentials, node, and cluster templates.

A fresh Kubernetes cluster is now only a few lines in our plan:

Example 5: Kubernetes Cluster

```
# Rancher cluster
resource "rancher2_cluster" "cluster_az" {
  name          = "az-${random_id.instance_id.hex}"
  description   = "Terraform"
  cluster_template_id = ...template_az.id
  cluster_template_revision_id = ...default_revision_id
  depends_on = [rancher2_cluster_template.template_az]
}
```

We give our cluster a name and a description and link it to the three templates we defined before.

We also need at least one node pool in our plan:

Example 6: Node Pool

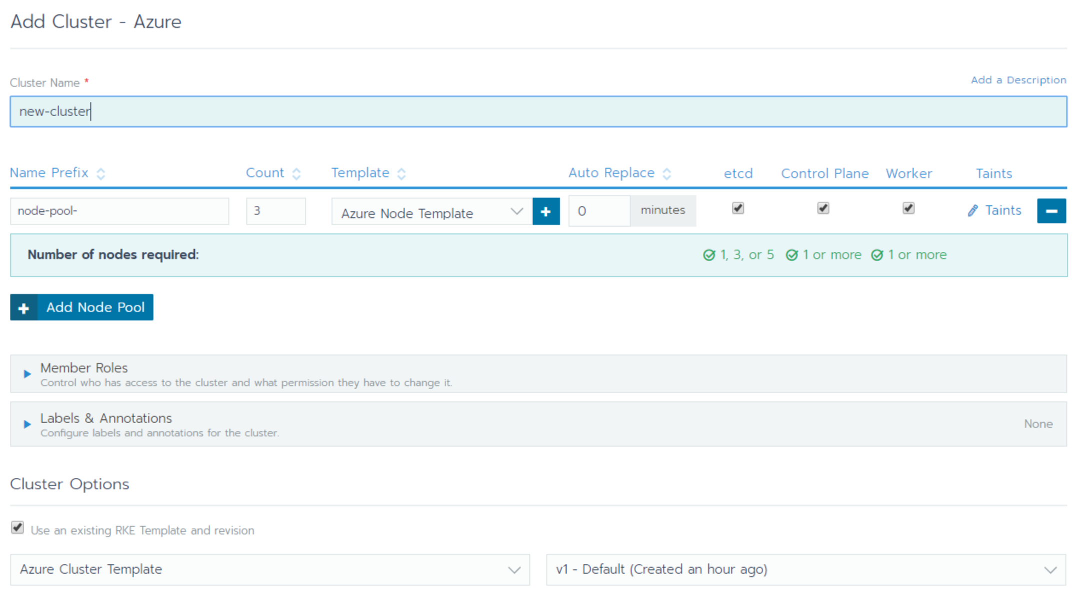
```
# Rancher node pool
resource "rancher2_node_pool" "nodepool_az" {
  cluster_id = rancher2_cluster.cluster_az.id
  name = "nodepool"
  hostname_prefix = "rke-${random_id.instance_id.hex}-"
  node_template_id = rancher2_node_template.template_az.id
  quantity = var.numnodes
  control_plane = true
  etcd = true
  worker = true
}
```

In this example, we define a single node pool with control plane and worker roles. Now we're done, and we can check the syntax of our plan files with `terraform plan`, and then execute the plan with `terraform apply`.

That's all that it takes!

In the Rancher GUI, creating a cluster is equally easy:

Figure 6: Cluster Creation



Add Cluster - Azure

Cluster Name * — Add a Description

new-cluster

Name Prefix	Count	Template	Auto Replace	etcd	Control Plane	Worker	Taints
node-pool-	3	Azure Node Template	0 minutes	✓	✓	✓	Taints

Number of nodes required: ✓ 1, 3, or 5 ✓ 1 or more ✓ 1 or more

Add Node Pool

Member Roles
Control who has access to the cluster and what permission they have to change it.

Labels & Annotations
Configure labels and annotations for the cluster. None

Cluster Options

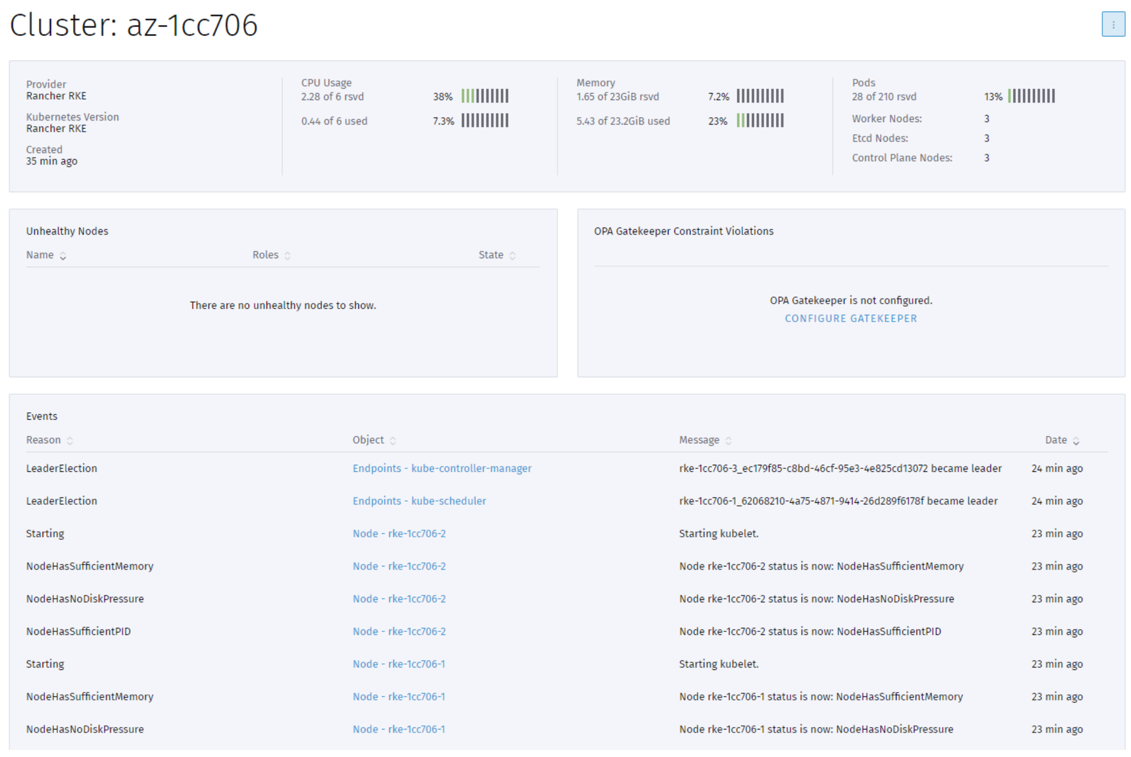
✓ Use an existing RKE Template and revision

Azure Cluster Template — v1 - Default (Created an hour ago)

With predefined templates, creating well-defined Kubernetes clusters is no longer a difficult task at all, and it does no longer require any in-depth knowledge of Kubernetes.

After successful creation, Rancher provides a convenient dashboard for the newly created cluster:

Figure 7: Rancher Dashboard



Cluster: az-1cc706

Provider Rancher RKE	CPU Usage 2.28 of 6 rsvd 38% 0.44 of 6 used 7.3%	Memory 1.65 of 23GiB rsvd 7.2% 5.43 of 23.2GiB used 23%	Pods 28 of 210 rsvd 13%
Kubernetes Version Rancher RKE			Worker Nodes: 3
Created 35 min ago			Etcd Nodes: 3
			Control Plane Nodes: 3

Unhealthy Nodes

Name	Roles	State

There are no unhealthy nodes to show.

OPA Gatekeeper Constraint Violations

OPA Gatekeeper is not configured.
CONFIGURE GATEKEEPER

Events

Reason	Object	Message	Date
LeaderElection	Endpoints - kube-controller-manager	rke-1cc706-3_ec179f85-c8bd-46cf-95e3-4e825cd13072 became leader	24 min ago
LeaderElection	Endpoints - kube-scheduler	rke-1cc706-1_62068210-4a75-4871-9414-26d289f6178f became leader	24 min ago
Starting	Node - rke-1cc706-2	Starting kubelet.	23 min ago
NodeHasSufficientMemory	Node - rke-1cc706-2	Node rke-1cc706-2 status is now: NodeHasSufficientMemory	23 min ago
NodeHasNoDiskPressure	Node - rke-1cc706-2	Node rke-1cc706-2 status is now: NodeHasNoDiskPressure	23 min ago
NodeHasSufficientPID	Node - rke-1cc706-2	Node rke-1cc706-2 status is now: NodeHasSufficientPID	23 min ago
Starting	Node - rke-1cc706-1	Starting kubelet.	23 min ago
NodeHasSufficientMemory	Node - rke-1cc706-1	Node rke-1cc706-1 status is now: NodeHasSufficientMemory	23 min ago
NodeHasNoDiskPressure	Node - rke-1cc706-1	Node rke-1cc706-1 status is now: NodeHasNoDiskPressure	23 min ago

This figure above shows the new Rancher dashboard layout, as introduced in March 2020 with Rancher 2.4.

4 CIS Scans

4.1 CIS Benchmarks for Kubernetes

In the previous chapter, we've covered the necessary mechanism to codify and enforce standards for our Kubernetai across our IT organization, using Terraform and Rancher templates. Now let's move one step further and look at checking our clusters for security issues.

There are many security standards and recommendations for IT, but there is one organization that's universally recognized as leading the field of IT security and compliance: The Center for Internet Security, Inc. (CIS®)

CIS is a community-driven nonprofit and responsible for CIS Controls® and CIS Benchmarks ™, which are globally recognized as the best practices for securing IT systems and data. If you want to look at the actual CIS Benchmarks, they are available for download on the CIS website; you'll need to register with CIS first, though.

Testing a platform against a CIS Benchmark is usually a lengthy process. Fortunately, there are several automated scan tools available for Kubernetes, and with Rancher 2.4, a CIS Scan is now available for all managed Clusters right from the Rancher GUI.

CIS Scans can be invoked manually or scheduled regularly, with Rancher's Alert Manager reporting it. With Terraform provider version 1.8.0 (March 2020), this feature is now also available in Terraform, too.[15]

4.2 CIS Scan GUI

In the classic GUI, at the cluster level, Rancher offers two choices of CIS Scans:

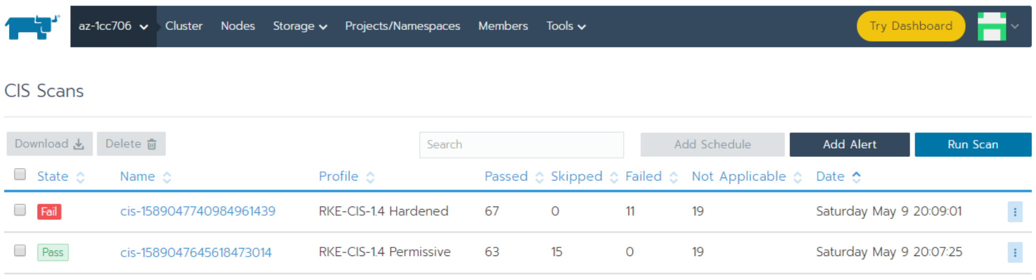
- RKE-CIS-1.4 Permissive
- RKE-CIS-1.4 Hardened

Both scans are based on the Kubernetes CIS Benchmark version 1.4, with different sets of enabled controls adapted by Rancher to the underlying Rancher Kubernetes Engine (RKE). A default installation will pass the permissive profile tests. To pass the hardened profile, you'll need to adhere fully to the Rancher 2.3 Hardening guide.[16]

The cluster we created in the previous chapter is using mostly default values and thus does not pass the scan with the hardened profile:

[15]See *Rancher Labs (2020)*: Changelog. [9]
[16]See *Rancher Labs (2019)*: Hardening Guide. [7]

Figure 8: CIS Scan GUI



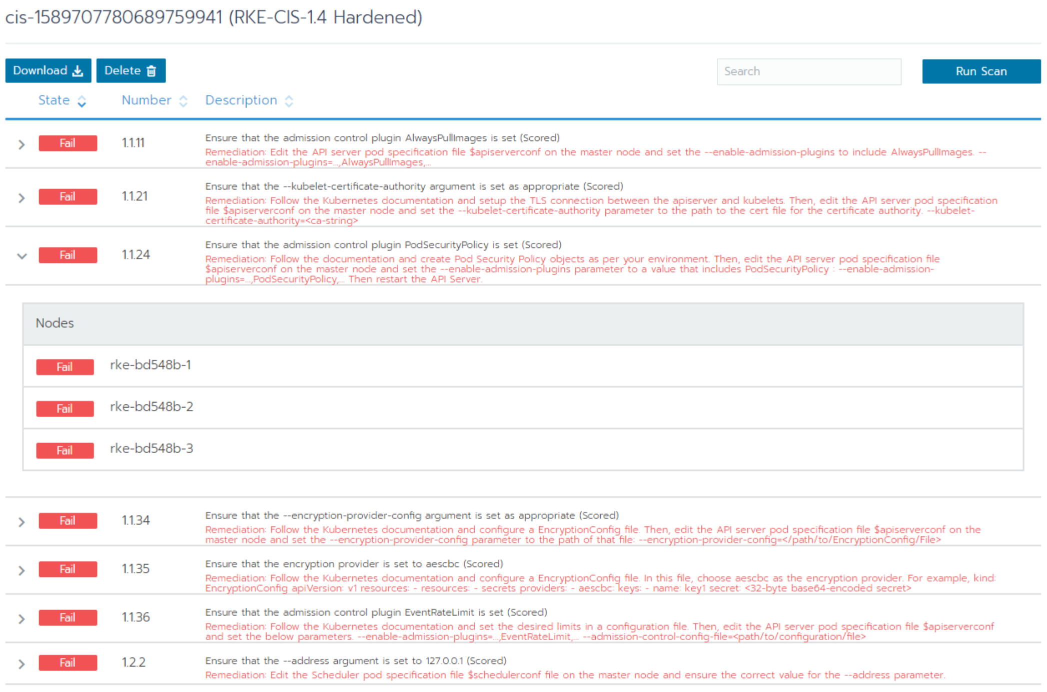
CIS Scans

State	Name	Profile	Passed	Skipped	Failed	Not Applicable	Date
Fail	cis-1589047740984961439	RKE-CIS-1.4 Hardened	67	0	11	19	Saturday May 9 20:09:01
Pass	cis-1589047645618473014	RKE-CIS-1.4 Permissive	63	15	0	19	Saturday May 9 20:07:25

4.3 Hardened CIS Scan

We will not go through all hardening steps in detail, but look at one control as an example:

Figure 9: Hardened CIS Scan



cis-1589707780689759941 (RKE-CIS-1.4 Hardened)

State	Number	Description
Fail	1.1.11	Ensure that the admission control plugin AlwaysPullImages is set (Scored) Remediation: Edit the API server pod specification file $apiserverconf on the master node and set the --enable-admission-plugins to include AlwaysPullImages. --enable-admission-plugins=...,AlwaysPullImages,...
Fail	1.1.21	Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored) Remediation: Follow the Kubernetes documentation and setup the TLS connection between the apiserver and kubelets. Then, edit the API server pod specification file $apiserverconf on the master node and set the --kubelet-certificate-authority parameter to the path to the cert file for the certificate authority. --kubelet-certificate-authority=<ca-string>
Fail	1.1.24	Ensure that the admission control plugin PodSecurityPolicy is set (Scored) Remediation: Follow the documentation and create Pod Security Policy objects as per your environment. Then, edit the API server pod specification file $apiserverconf on the master node and set the --enable-admission-plugins parameter to a value that includes PodSecurityPolicy : --enable-admission-plugins=...,PodSecurityPolicy,... Then restart the API Server.

Nodes

Fail	rke-bd548b-1
Fail	rke-bd548b-2
Fail	rke-bd548b-3

State	Number	Description
Fail	1.1.34	Ensure that the --encryption-provider-config argument is set as appropriate (Scored) Remediation: Follow the Kubernetes documentation and configure a EncryptionConfig file. Then, edit the API server pod specification file $apiserverconf on the master node and set the --encryption-provider-config parameter to the path of that file: --encryption-provider-config=</path/to/EncryptionConfig/File>
Fail	1.1.35	Ensure that the encryption provider is set to aescbc (Scored) Remediation: Follow the Kubernetes documentation and configure a EncryptionConfig file. In this file, choose aescbc as the encryption provider. For example, kind: EncryptionConfig apiVersion: v1 resources: - resources: - secrets providers: - aescbc: keys: - name: key1 secret: <32-byte base64-encoded secret>
Fail	1.1.36	Ensure that the admission control plugin EventRateLimit is set (Scored) Remediation: Follow the Kubernetes documentation and set the desired limits in a configuration file. Then, edit the API server pod specification file $apiserverconf and set the below parameters. --enable-admission-plugins=...,EventRateLimit,... --admission-control-config-file=<path/to/configuration/file>
Fail	1.2.2	Ensure that the --address argument is set to 127.0.0.1 (Scored) Remediation: Edit the Scheduler pod specification file $schedulerconf file on the master node and ensure the correct value for the --address parameter.

The control we'll be focusing on is 1.1.24: Ensure that the admission control plugin PodSecurityPolicy is set.

To fully harden your cluster, please follow the hardening instructions mentioned above; for now, we'll look at the missing Pod Security Policy.

4.4 Kubernetes Security Policies

What is a Pod Security Policy (PSP)? Kubernetes provides two types of security policies, one for pod security and one for network security. Pod security policies, as the name implies, govern security-relevant aspects of pod specification,[17], whereas network policies govern the allowed communication between groups of pods and the outside world.[18]

It is good practice when running a Kubernetes cluster in production to secure access with Pod Security Policies.[19] Rancher offers two pre-built PSPs (named "restricted" and "unrestricted") and a GUI to create your own.[20]

There are many controls within a Pod Security Policy that we won't cover in this document; instead, we'll focus on the admission controller.

4.5 Remediation

The scan gives us the following remediation instruction:

```
Remediation: Follow the documentation and create Pod
Security Policy objects as per your environment
```

To mitigate the failed control, what we need to do is to add a default Pod Security Policy. Fortunately, that's relatively easy in Rancher. In the cluster template, we need to enable PSP support and define the default policy.

We can do that by adding the following line to our template definition in Terraform:

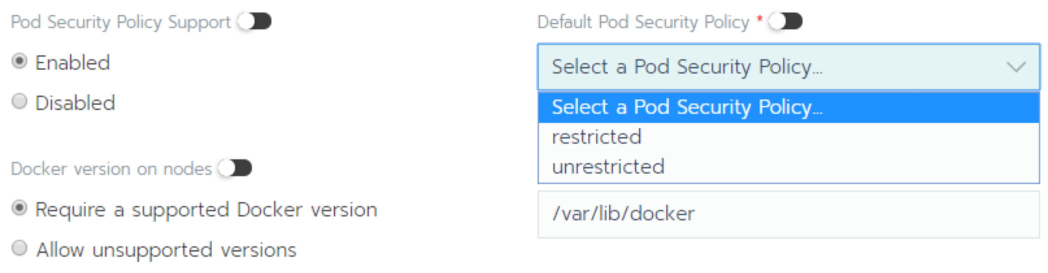
Example 7: Cluster Template with PSP

```
default_pod_security_policy_template_id = "restricted"
```

Or, if your prefer the Rancher GUI, you can set the default Pod Security Policy there too:

[17]See *The Linux Foundation (2019)*: Pod Security Policies. [16]
[18]See *The Linux Foundation (2019)*: Network Policies. [15]
[19]See *Price, J. (2020)*: Kubernetes - Pod Security Policies. [6]
[20]See *Iradier, A. (2020)*: Enhancing Kubernetes Security with Pod Security Policies. [4]

Figure 10: Rancher PSP Support



Pod Security Policy Support

[✓ Enabled] Disabled

Docker version on nodes

[✓ Require a supported Docker version] Allow unsupported versions

Default Pod Security Policy *

Select a Pod Security Policy...

Once you've enabled the PSP, the control will pass in the CIS scan. It's an iterative process: Based on your IT organization's needs, you'll have to identify the controls critical for your business and implement the remediation steps one by one, as needed, to harden your cluster.

Automated CIS scans will then give you a valuable tool to check all your Kubernetai regularly for security issues and regulatory compliance, and help you to act on issues accordingly.

5 Summary and recommendations

Infrastructure-as-Code is the perfect tool to enforce regulatory compliance and security hardening uniformly across all deployed Kubernetes clusters.

Terraform provides the declarative definition for the infrastructure and Rancher, through templates and access control, the necessary controls for the installation and configuration of Kubernetes itself.

The Center of Internet Security offers benchmarks to test Kubernetes cluster against. Rancher has CIS Scans integrated and provides the ability to mitigate the findings.

We were able to show that the combination of Terraform, together with Rancher templates, provides an ideal solution for Enterprise IT to provision a secure and compliant container run-time environment and manage the infrastructure life cycle. By having all infrastructure well defined and under revision control and all infrastructure deployments automated, you can easily apply GRC controls.

There are alternatives to this combination. Platform-native solutions for the major hyperscale platforms do exist, for example, ARM templates on Microsoft Azure or Cloud Formation on Amazon Web Services. Also, the leading on-premise virtualization platforms, VMware and Hyper-V, each have a native orchestration solution.

Rancher and Terraform, however, are an open-source and provider-independent option and thus the recommended choice; both tools have been around for a couple of years and amassed a large and mostly loyal user base.

There are other open-source tools, such as the very recently released CDK8S by Amazon Web Services. In my opinion, Rancher and Terraform are the most comprehensive and are ideally suited for cloud computing.

Infrastructure automation is by no means a new invention. Procedural tools like Ansible, Chef, or Operations Orchestration have been around for decades.

New developments in the area of serverless computing threaten computing infrastructure as a distinct category itself and might obliterate the need for secure and compliant orchestration tools, though.

Regardless of future developments, governance, risk management, and regulatory compliance will remain a key topic in Enterprise IT.

You can find all example plan files on my GitHub.

Happy Ranching!

References

[1] C. Frank. (2020) Deploy kubernetes clusters on microsoft azure with rancher. [Access 2020-05-10]. [Online]. Available: https://rancher.com/blog/2020/build-kubernetes-clusters-on-azure

[2] C. Frank. (2020) Multi-cluster management fuer containerumgebungen. [Access 2020-05-10]. [Online]. Available: https://storage.googleapis.com/bucket.chfrank.net/Managing%20multiple%20clusters%20for%20container%20run-time%20environments.pdf

[3] HashiCorp. (2020) Deliver infrastructure as code with terraform. [Access 2020-05-10]. [Online]. Available: https://www.terraform.io/

[4] A. Iradier. (2020) Enhancing kubernetes security with pod security policies. [Access 2020-05-10]. [Online]. Available: https://rancher.com/blog/2020/pod-security-policies-part-2

[5] P. Mell and T. Grance. (2011) The nist definition of cloud computing. [Access 2020-05-10]. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-145

[6] J. Price. (2020) Kubernetes - pod security policies. [Access 2020-05-10]. [Online]. Available: https://developer.squareup.com/blog/kubernetes-pod-security-policies/

[7] Rancher Labs. (2019) Hardening guide - rancher v2.3.x. [Access 2020-05-10]. [Online]. Available: https://rancher.com/docs/rancher/v2.x/en/security/hardening-2.3/

[8] Rancher Labs. (2019) Introducing the rancher 2 terraform provider. [Access 2020-05-10]. [Online]. Available: https://rancher.com/blog/2019/rancher-2-terraform-provider/

[9] Rancher Labs. (2020) Rancher 2 terraform provider 1.8.3. [Access 2020-05-10]. [Online]. Available: https://github.com/terraform-providers/terraform-provider-rancher2/blob/master/CHANGELOG.md

[10] Rancher Labs. (2020) Rancher 2.4 technical architecture. [Access 2020-05-10]. [Online]. Available: https://info.rancher.com/rancher2-technical-architecture

[11] Rancher Labs. (2020) Run kubernetes everywhere. [Access 2020-05-10]. [Online]. Available: https://rancher.com/

[12] A. Saguy and J. Williams. (2020) Why we should all use they/them pronouns.

[Access 2020-05-10]. [Online]. Available: https://blogs.scientificamerican.com/voices/why-we-should-all-use-they-them-pronouns/

[13] P. Shankar. (2020) Runtime security in rancher with falco. [Access 2020-05-10]. [Online]. Available: https://rancher.com/blog/2020/runtime-security-with-falco

[14] M. Souppaya, J. Morello, and K. Scarfone. (2017) Application container security guide. [Access 2020-05-10]. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-190

[15] The Linux Foundation. (2020) Network policies. [Access 2020-05-10]. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/network-policies/

[16] The Linux Foundation. (2020) Pod security policies. [Access 2020-05-10]. [Online]. Available: https://kubernetes.io/docs/concepts/policy/pod-security-policy/

[17] The Linux Foundation. (2020) Production-grade container orchestration. [Access 2020-05-10]. [Online]. Available: https://kubernetes.io/

[18] D. Weibel. (2020) Architecting kubernetes clusters - how many should you have? [Access 2020-05-10]. [Online]. Available: https://learnk8s.io/how-many-clusters

[19] Y. Weizman. (2020) Threat matrix for kubernetes. [Access 2020-05-10]. [Online]. Available: https://www.microsoft.com/security/blog/2020/04/02/attack-matrix-kubernetes/