

<https://www.youtube.com/watch?v=TS-7x9Zn-Yk>

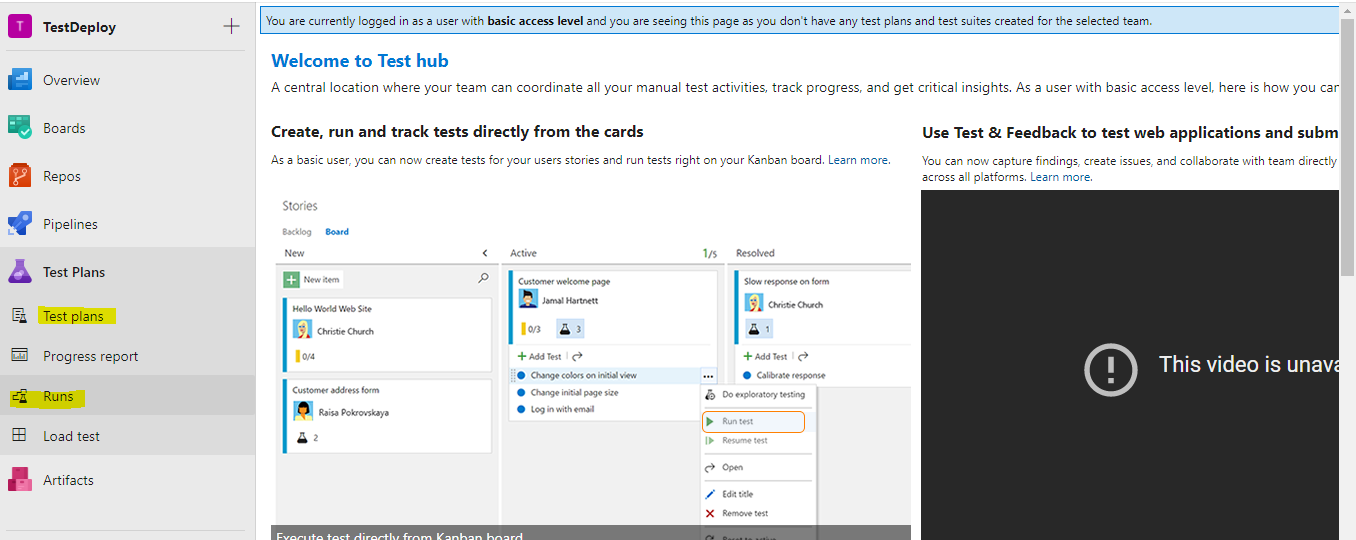
<https://www.youtube.com/watch?v=MoFAR_6KDFo>

<https://www.youtube.com/watch?v=vBx7WY25fM0>

<https://www.c-sharpcorner.com/article/event-handling-in-net-using-C-Sharp/>

<https://docs.microsoft.com/en-us/azure/event-grid/event-handlers>

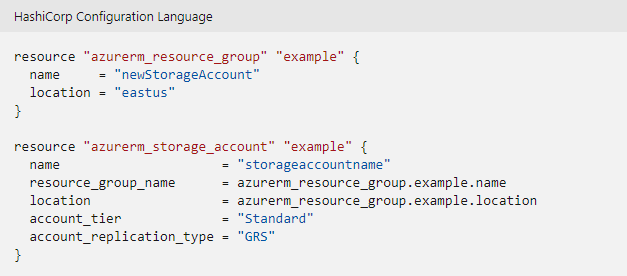
TESTS:



# Terraform with Azure

# [Hashicorp Terraform](https://www.terraform.io/) is an open-source tool for provisioning and managing cloud infrastructure. It codifies infrastructure in configuration files that describe the topology of cloud resources. These resources include virtual machines, storage accounts, and networking interfaces. The Terraform CLI provides a simple mechanism to deploy and version the configuration files to Azure.

*Terraform provider for Azure is an abstraction on top of Azure APIs. This abstraction is beneficial because the API complexities are obfuscated.*

**

* Lowers the potential for human errors while deploying and managing infrastructure.
* Deploys the same template multiple times to create identical development, test, and production environments.
* Reduces the cost of development and test environments by creating them on-demand.

IaC Approaches

IaC can be declarative or imperative:

·       **Declarative** syntax makes up more of a functional deployment. The final state is declared in a way to define *“what”* state the deployment should be in. As a result, the environment gets configured in a declared state as the result. A perfect example of declarative syntax can be found by using ARM Templates.

·       **Imperative** syntax is a bit different. Imperative syntax is more procedural in comparison to declarative syntax. The code you write, runs through each configuration step-by-step. Imperative syntax is the process of *“how”* the system is to be configured + what steps need to be taken (in the exact order) to reach the final state. A perfect example of imperative syntax can be found by using PowerShell (of the non-desired state configuration flavor).

Is ARM declarative or imperative?

The ARM Template is an example of **declarative syntax**



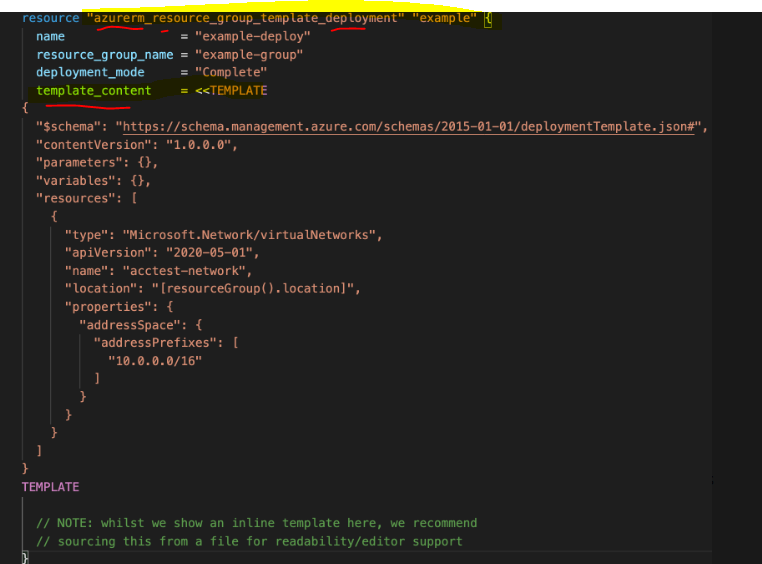
There are many tools that can help tackle infrastructure as code and they range from cloud-specific tools like [Azure Resource Manager Templates](https://docs.microsoft.com/azure/azure-resource-manager/templates/?WT.mc_id=devops-9648-zdeptawa) to open-source cloud or host agnostic tools like [HashiCorp Terraform](https://azure.microsoft.com/solutions/devops/terraform/?WT.mc_id=devops-9648-zdeptawa).

***Which is right for you?***  
If you want to have your infrastructure span several cloud providers or platforms using these tools alone, you will need to work with each tool from each provider – and each cloud provider’s tooling will use different syntax and verbiage. This is where cloud or host agnostic tools like [HashiCorp Terraform](https://azure.microsoft.com/solutions/devops/terraform/?WT.mc_id=devops-9648-zdeptawa) can really make life easier working across clouds as well as on-premises environments.

No matter the provider, all HashiCorp Terraform configuration files use [HCL (HashiCorp Configuration Language)](https://www.terraform.io/docs/configuration/syntax.html) syntax to define resources. This means the syntax for environments that span several different providers will be the same, and the process to deploy or make changes with that code is identical as well.

How to use existing ARM template in terraform?

### [HashiCorp Terraform](https://azure.microsoft.com/solutions/devops/terraform/?WT.mc_id=devops-9648-zdeptawa) has a resource called azurerm\_resource\_group\_template\_deployment for the AzureRM provider. This resource allows you to deploy an [ARM Template](https://docs.microsoft.com/azure/azure-resource-manager/templates/?WT.mc_id=devops-9648-zdeptawa) through [HashiCorp Terraform](https://azure.microsoft.com/solutions/devops/terraform/?WT.mc_id=devops-9648-zdeptawa). This means you can use the same [ARM Template](https://docs.microsoft.com/azure/azure-resource-manager/templates/?WT.mc_id=devops-9648-zdeptawa) code within your [HashiCorp Terraform](https://azure.microsoft.com/solutions/devops/terraform/?WT.mc_id=devops-9648-zdeptawa) configuration file without the need to translate the [ARM Template](https://docs.microsoft.com/azure/azure-resource-manager/templates/?WT.mc_id=devops-9648-zdeptawa) into the various Terraform AzureRM resources that would be required to move from [ARM templates](https://docs.microsoft.com/azure/azure-resource-manager/templates/?WT.mc_id=devops-9648-zdeptawa) to [Terraform](https://azure.microsoft.com/solutions/devops/terraform/?WT.mc_id=devops-9648-zdeptawa).



### Configure a Linux VM with infrastructure in Azure using Terraform

. Implement the Terraform code  
Create a file named main.tf and insert the following code

# Configure the Microsoft Azure Provider

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "~>2.0"

}

}

}

provider "azurerm" {

features {}

}

# Create a resource group if it doesn't exist

resource "azurerm\_resource\_group" "myterraformgroup" {

name = "myResourceGroup"

location = "eastus"

tags = {

environment = "Terraform Demo"

}

}

# Create virtual network

resource "azurerm\_virtual\_network" "myterraformnetwork" {

name = "myVnet"

address\_space = ["10.0.0.0/16"]

location = "eastus"

resource\_group\_name = azurerm\_resource\_group.myterraformgroup.name

tags = {

environment = "Terraform Demo"

}

}

….etc….

To initialize the Terraform deployment, run [terraform init](https://www.terraform.io/docs/commands/init.html) : terraform init

After initialization, you create an execution plan by running [terraform plan](https://www.terraform.io/docs/commands/plan.html): terraform plan -out main.tfplan

* *The terraform plan command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.*
* *The optional -out parameter allows you to specify an output file for the plan. Using the -out parameter ensures that the plan you reviewed is exactly what is applied*.

### Once you're ready to apply the execution plan to your cloud infrastructure, you run [terraform apply](https://www.terraform.io/docs/commands/apply.html)

terraform apply main.tfplan

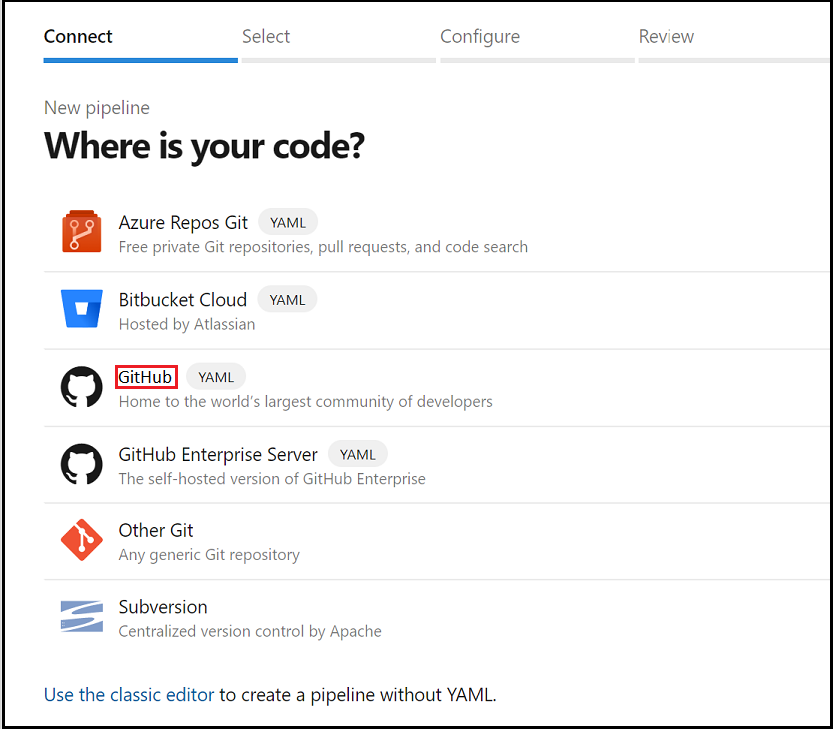
* *The terraform apply command above assumes you previously ran terraform plan -out main.tfplan.*
* *If you specified a different filename for the -out parameter, use that same filename in the call to terraform apply.*
* *If you didn't use the -out parameter, simply call terraform apply without any parameters*

## Automate integration tests using Azure Pipeline

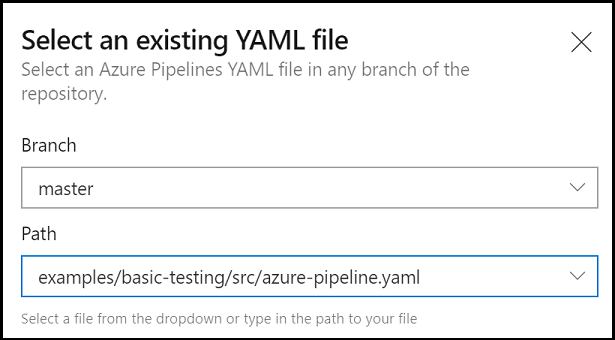
Continuous integration involves testing an entire system when a change is introduced. In this section, you see an Azure Pipeline configuration used to implement continuous integration

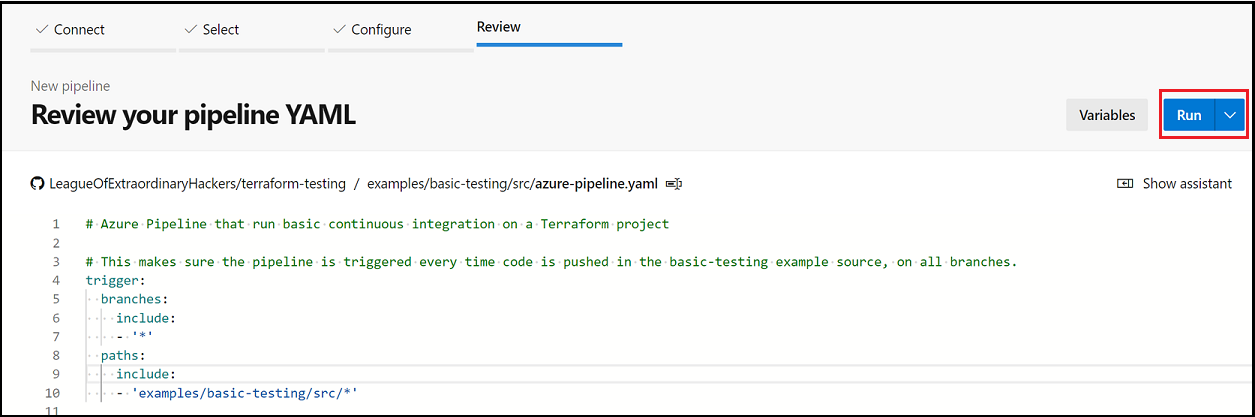
<https://github.com/Azure/terraform/blob/master/samples/integration-testing/src/azure-pipeline.yaml>

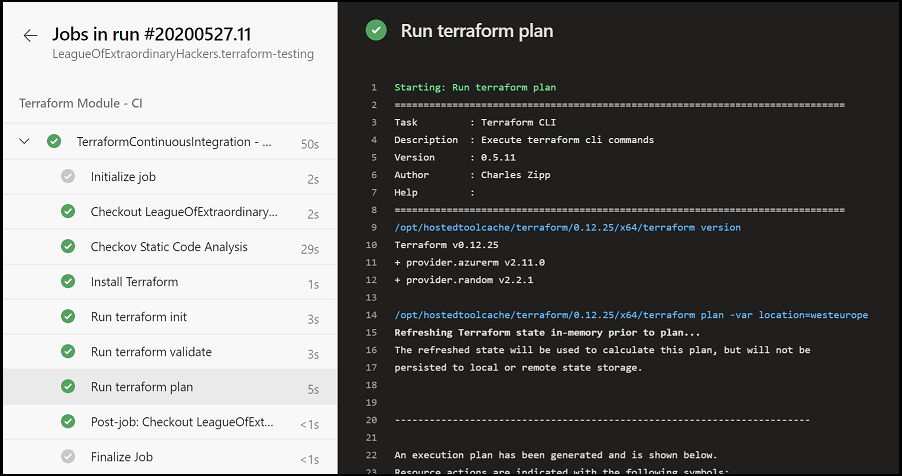


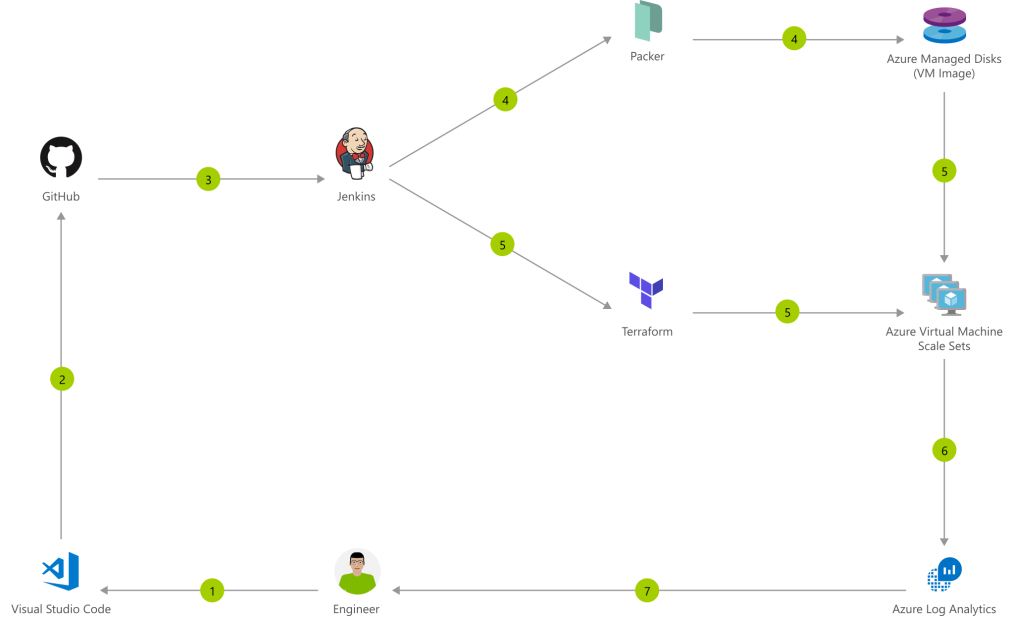












### Chef

Chef is an automation platform that helps define how your infrastructure is configured, deployed, and managed. Additional components included Chef Habitat for application lifecycle automation rather than the infrastructure and Chef InSpec that helps automate compliance with security and policy requirements. Chef Clients are installed on target machines, with one or more central Chef Servers that store and manage the configurations.

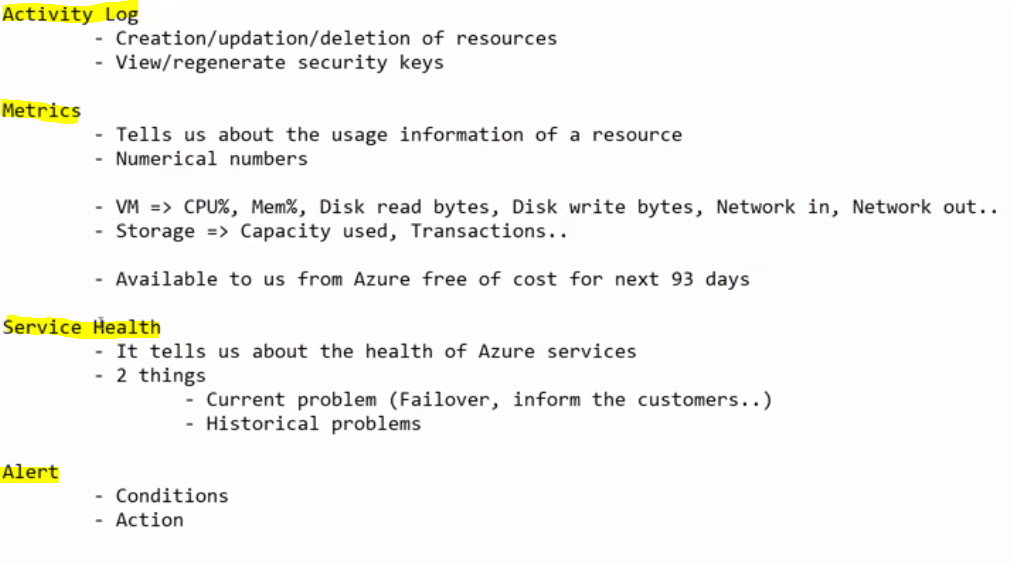
**Learn more**

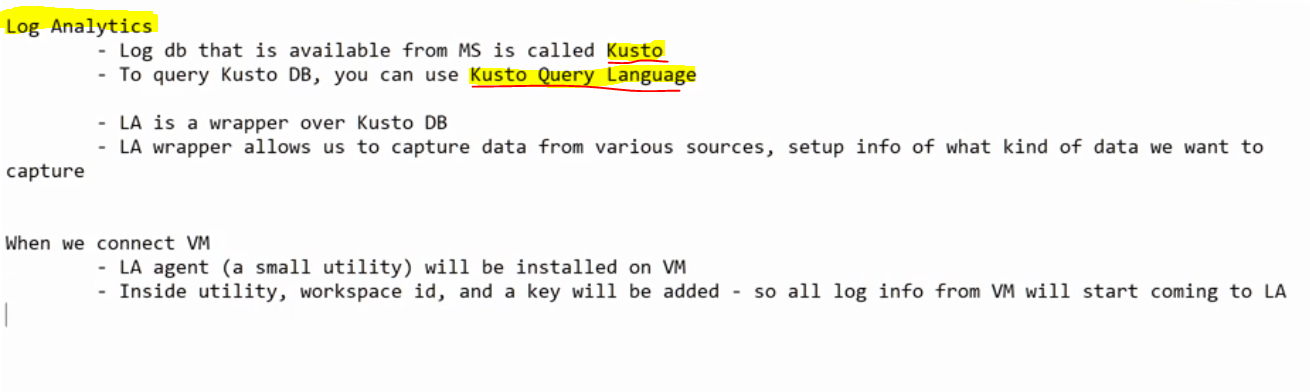
* [Documentation: An Overview of Chef](https://docs.chef.io/chef_overview.html)

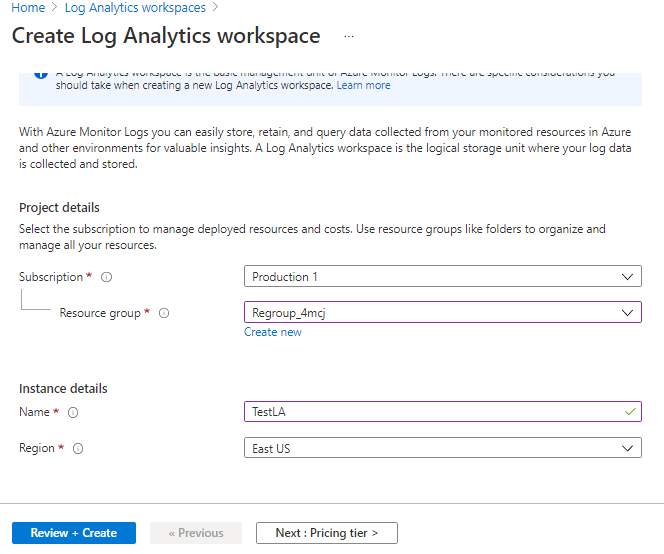
### Puppet

Puppet is an enterprise-ready automation platform that handles the application delivery and deployment process. Agents are installed on target machines to allow Puppet Master to run manifests that define the desired configuration of the Azure infrastructure and virtual machines. Puppet can integrate with other solutions such as Jenkins and GitHub for an improved DevOps workflow.

# 







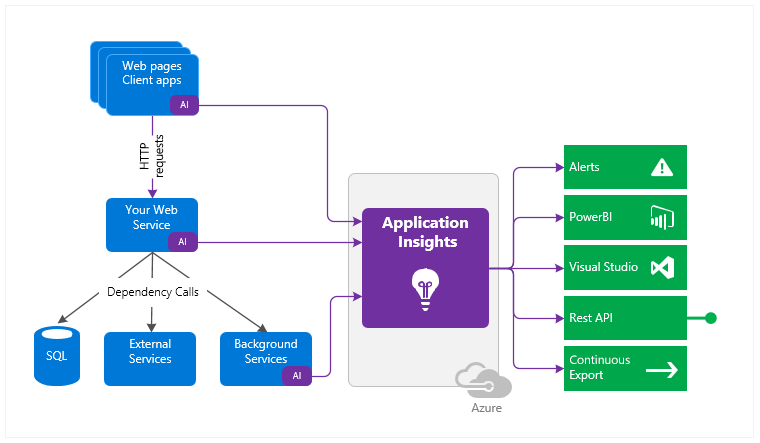
# Application Insights

<https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview>

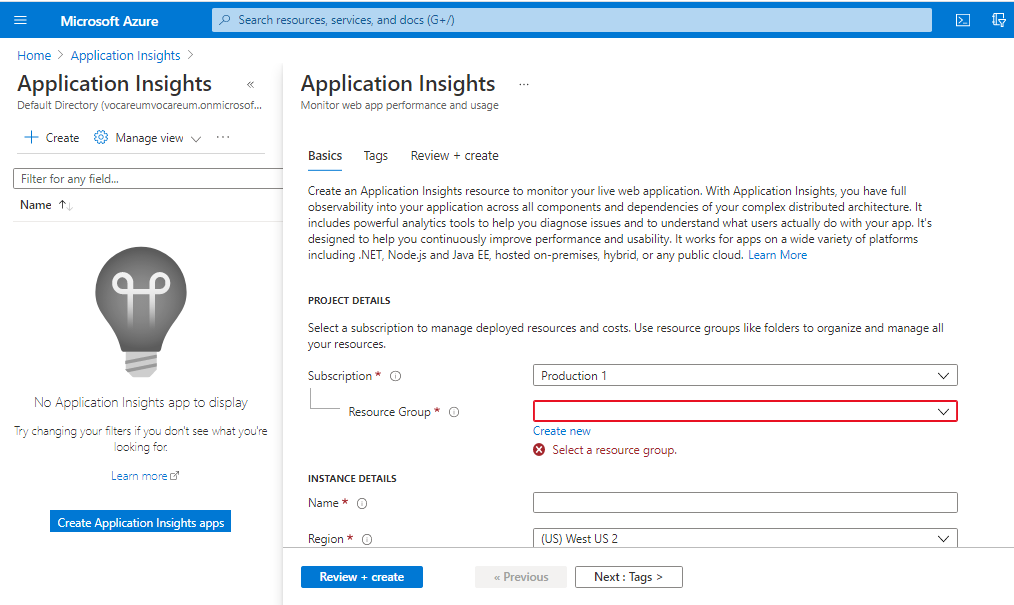
, a feature of [Azure Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/overview), is an extensible Application Performance Management (APM) service for developers and DevOps professionals

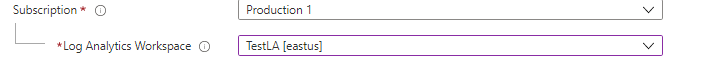
it to monitor your live applications. It will automatically detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability

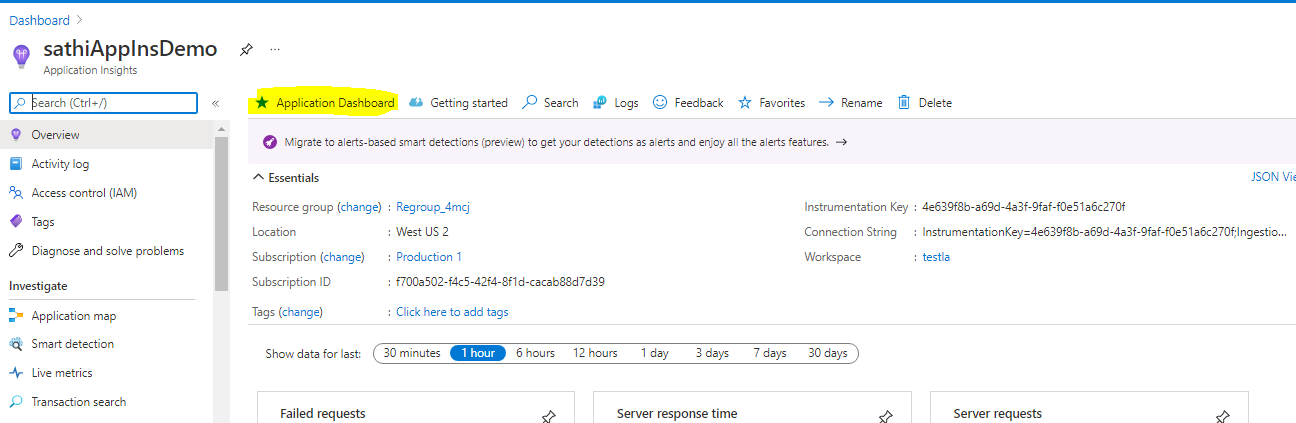


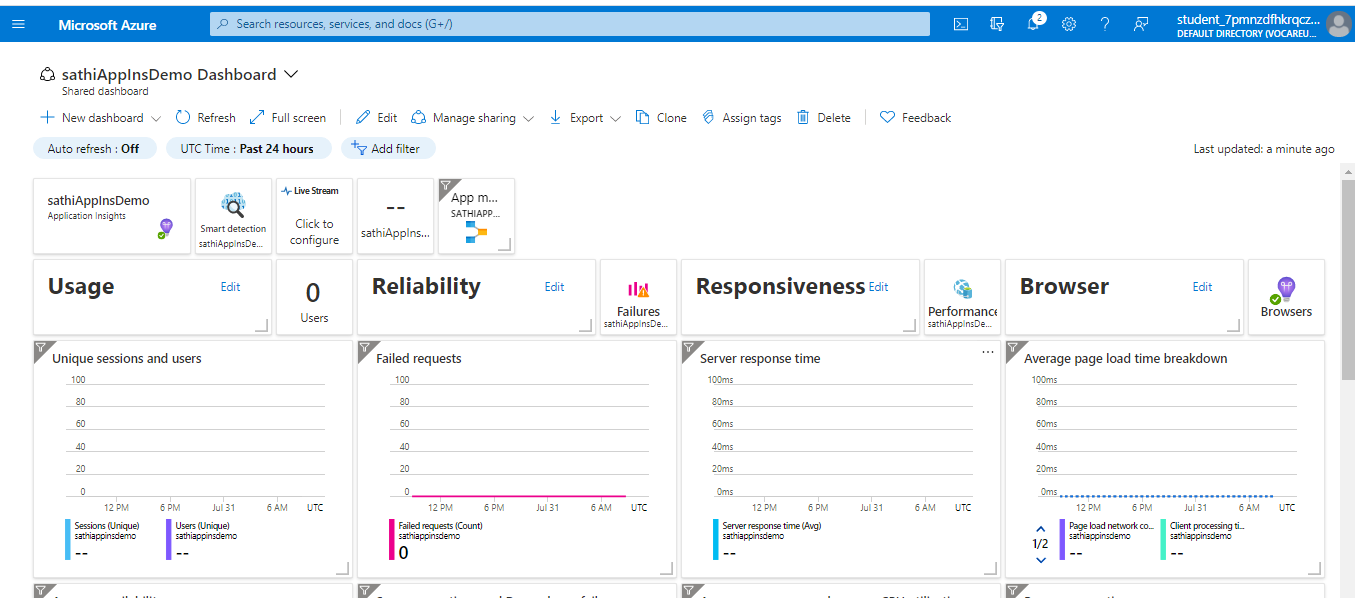


Monitor is free and available for around only 93 days history , Log Analytics needs to be used if we need to have data for more then 93 days



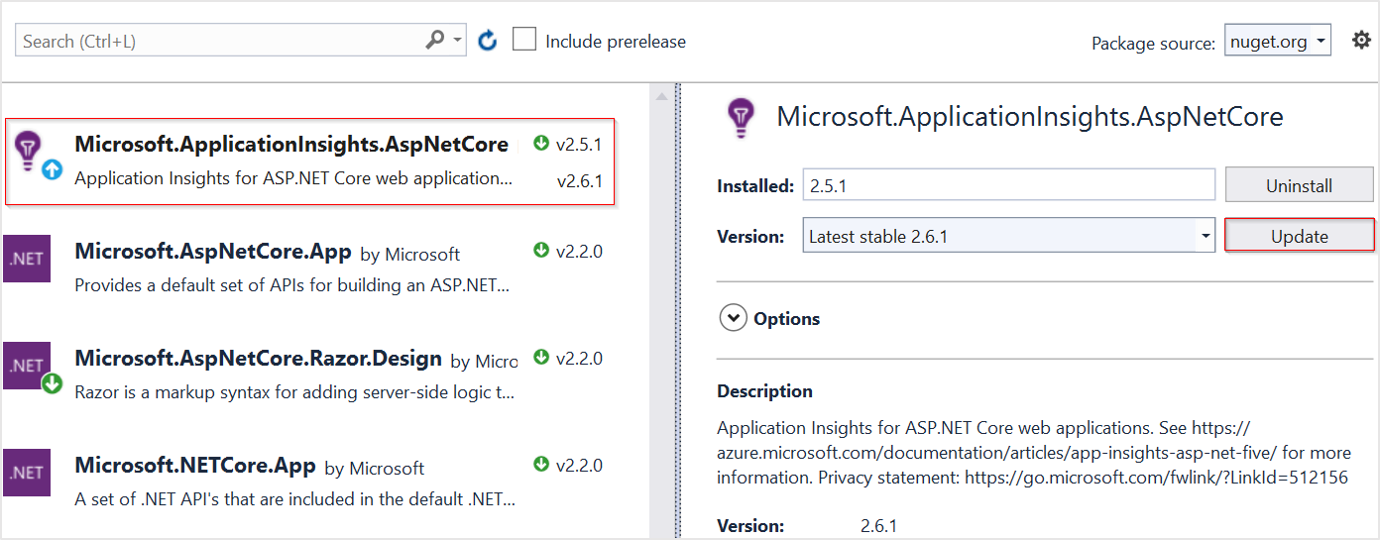




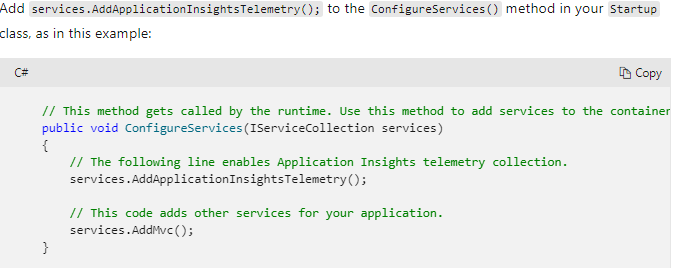


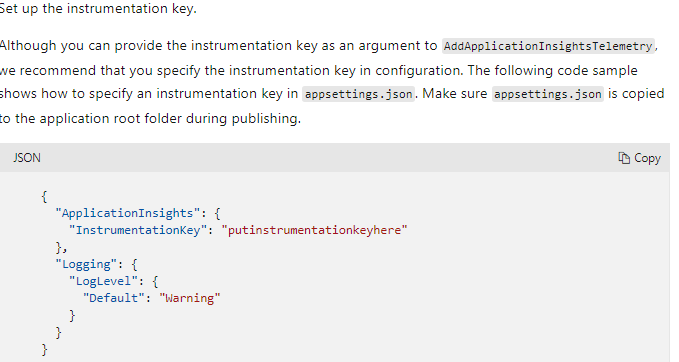
# Application Insights for ASP.NET Core applications

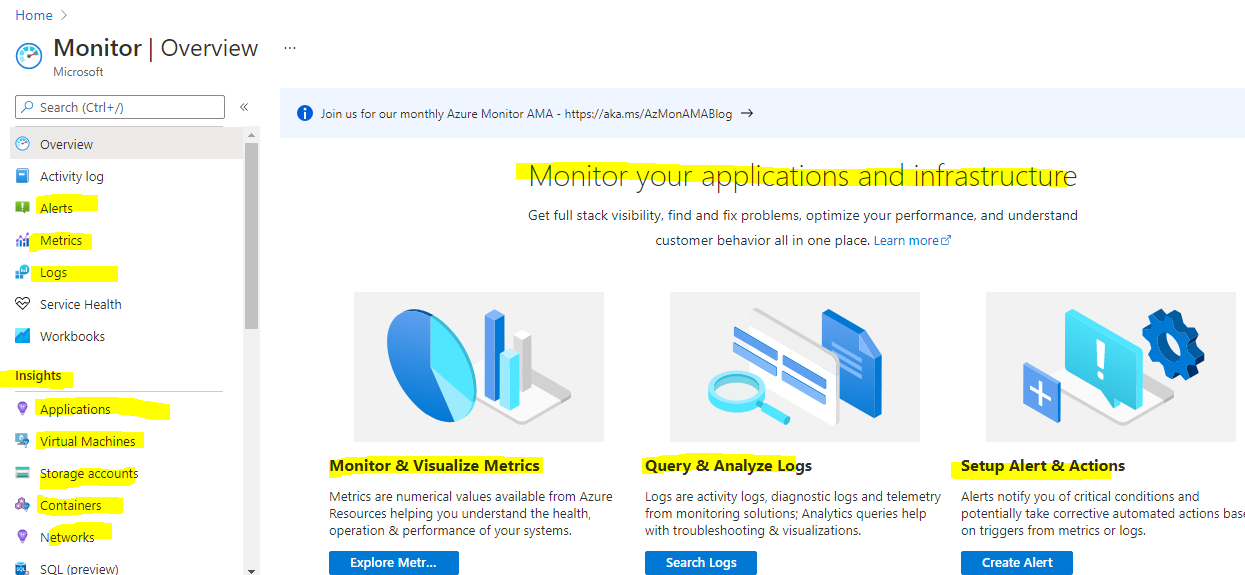
<https://docs.microsoft.com/en-us/azure/azure-monitor/app/asp-net-core>







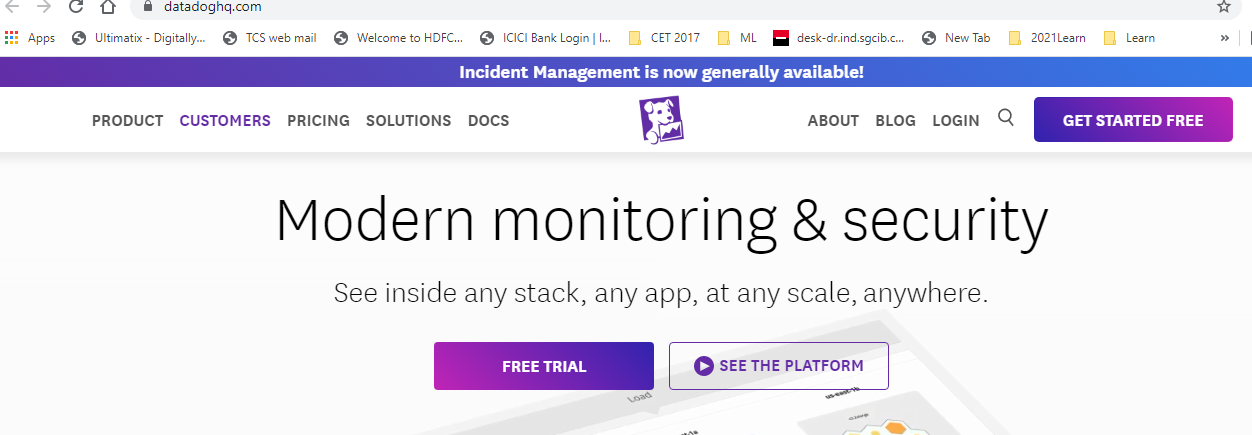




What is Datadog monitoring?

Datadog is a **monitoring and analytics tool for information technology (IT) and DevOps teams** that can be used to determine performance metrics as well as event monitoring for infrastructure and cloud services. The software can monitor services such as servers, databases and tools. Sends alerts







Microsoft Azure enables customers to migrate and modernize their applications to run in the cloud, in coordination with many partner solutions. One such partner is [Datadog](https://www.datadoghq.com/),

vailable via the [Azure Marketplace](https://azuremarketplace.microsoft.com/en-us/marketplace/), this solution provides a seamless experience for using the Datadog’s cloud monitoring solution in Azure.

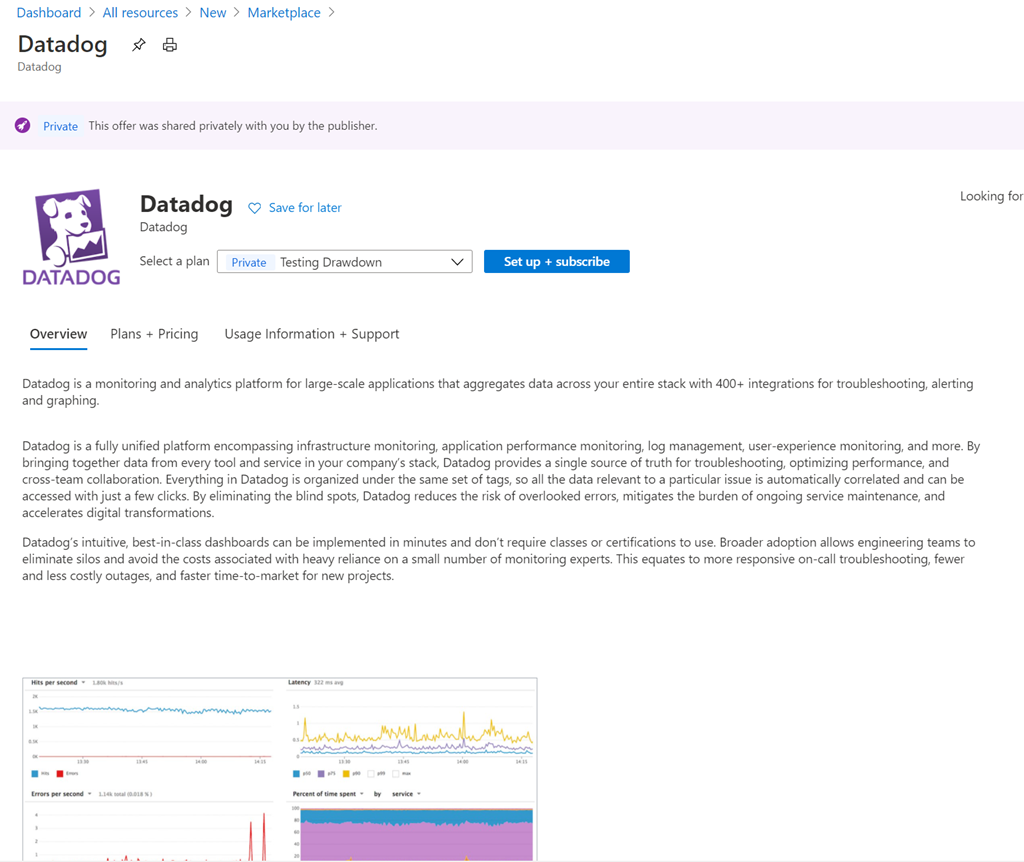
With the new Azure integration with Datadog, organizations can now fully map their legacy and cloud-based systems, monitoring real-time data during every phase of the cloud transition, and ensure that migrated applications meet performance targets. This integration combines Azure’s global presence, flexibility, security, and compliance with Datadog's logging and monitoring capabilities to create the best experience for enterprises

## Acquire and setup the Datadog solution

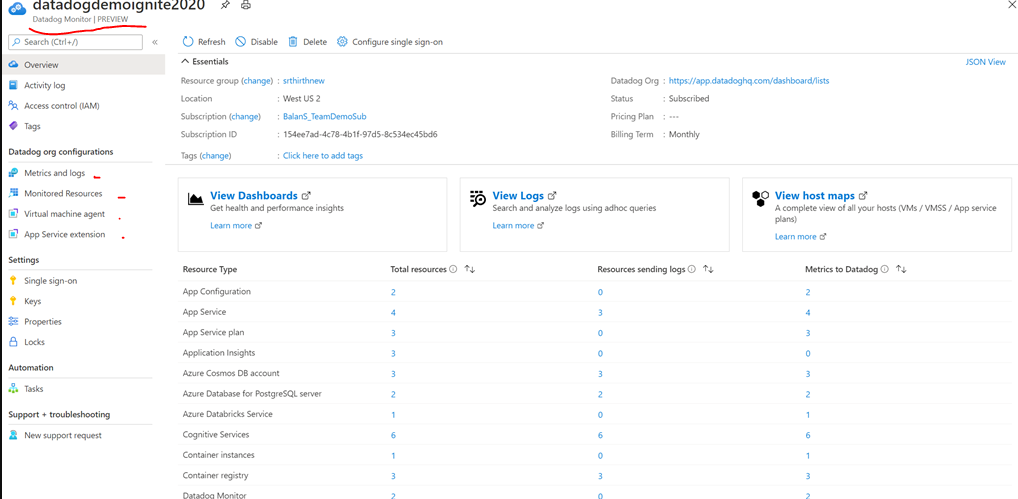
<https://azure.microsoft.com/en-in/blog/new-datadog-integration-with-azure-offers-a-seamless-configuration-experience/>

Now let’s follow the step-by-step process to acquire and setup the Datadog solution:

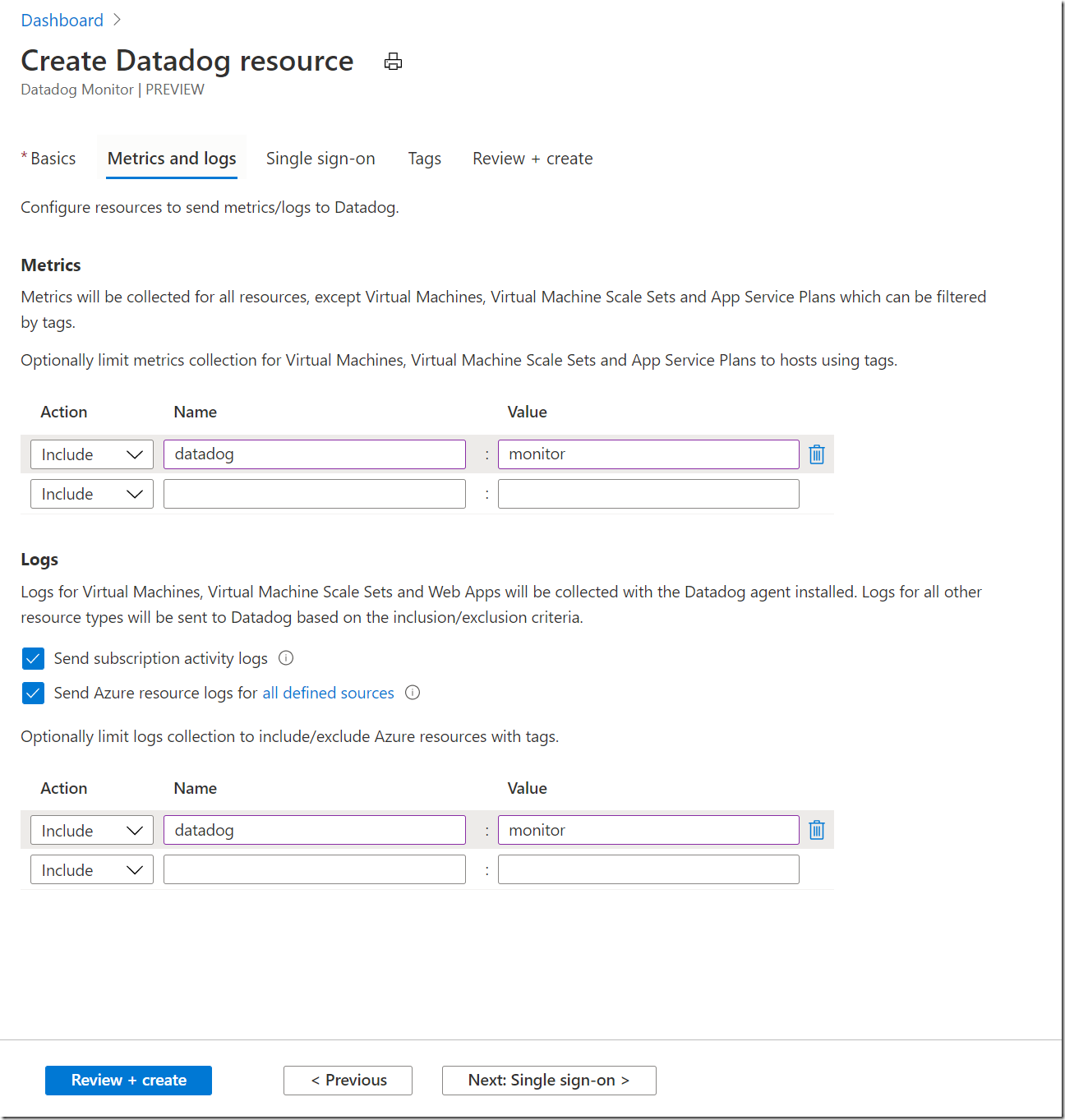
**Procuring the Datadog app**: Azure customers can procure the Datadog app through the Azure Marketplace.



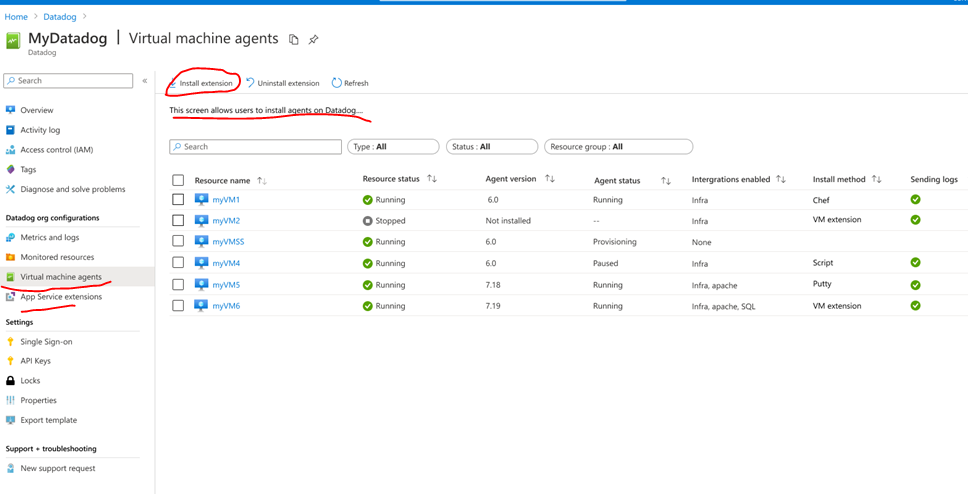
**Provisioning in the Azure portal:** After procuring in Azure Marketplace, customers can seamlessly provision Datadog as an integrated service on Azure via the Azure portal.



**Configuring logs and metrics:** Customers create a Datadog resource in Azure and configure which Azure resources send logs and metrics and to Datadog.

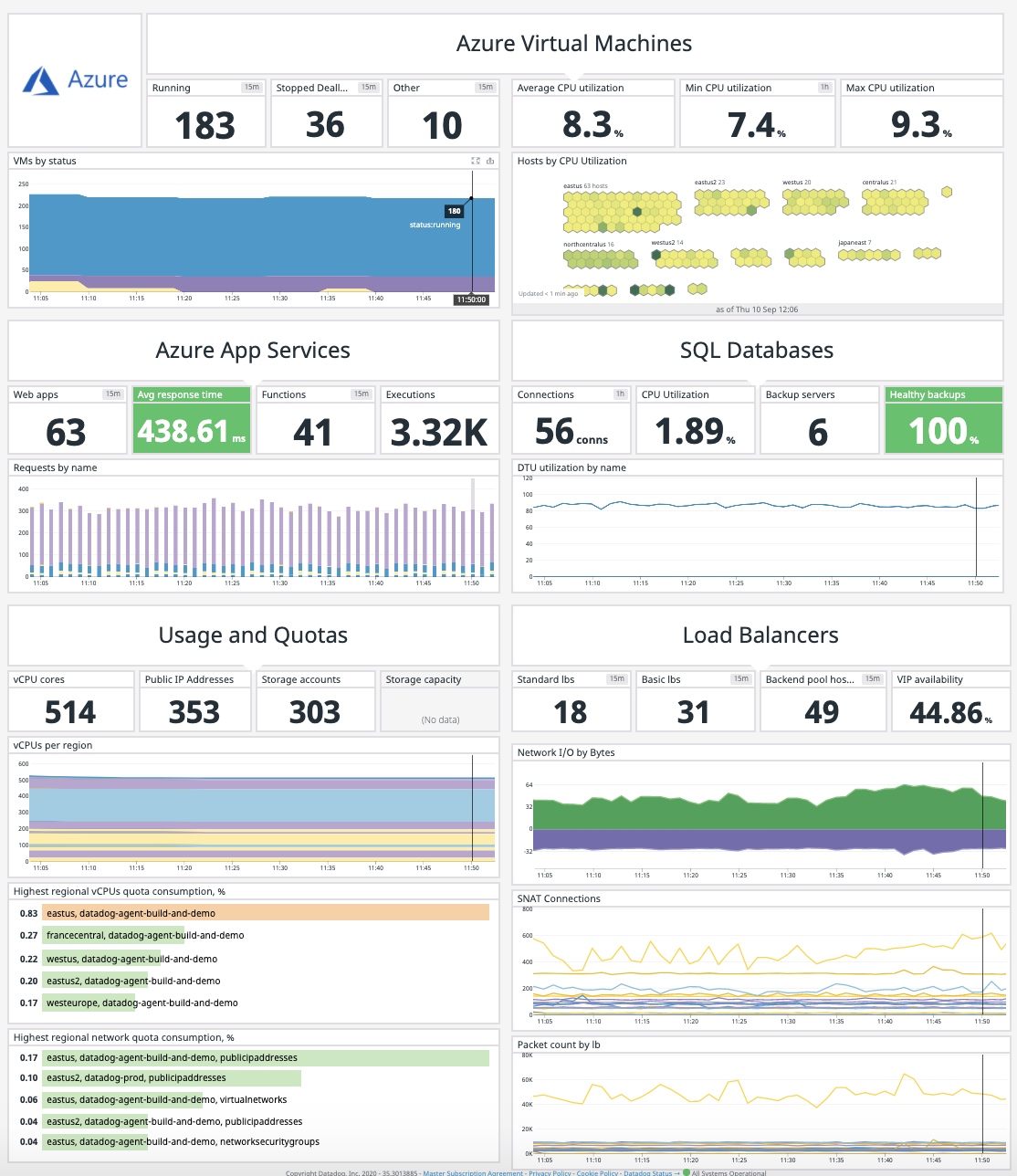


**Installing Datadog agent**: Customers can install the Datadog agent as an extension on virtual machines (VMs) and app services with a single click.



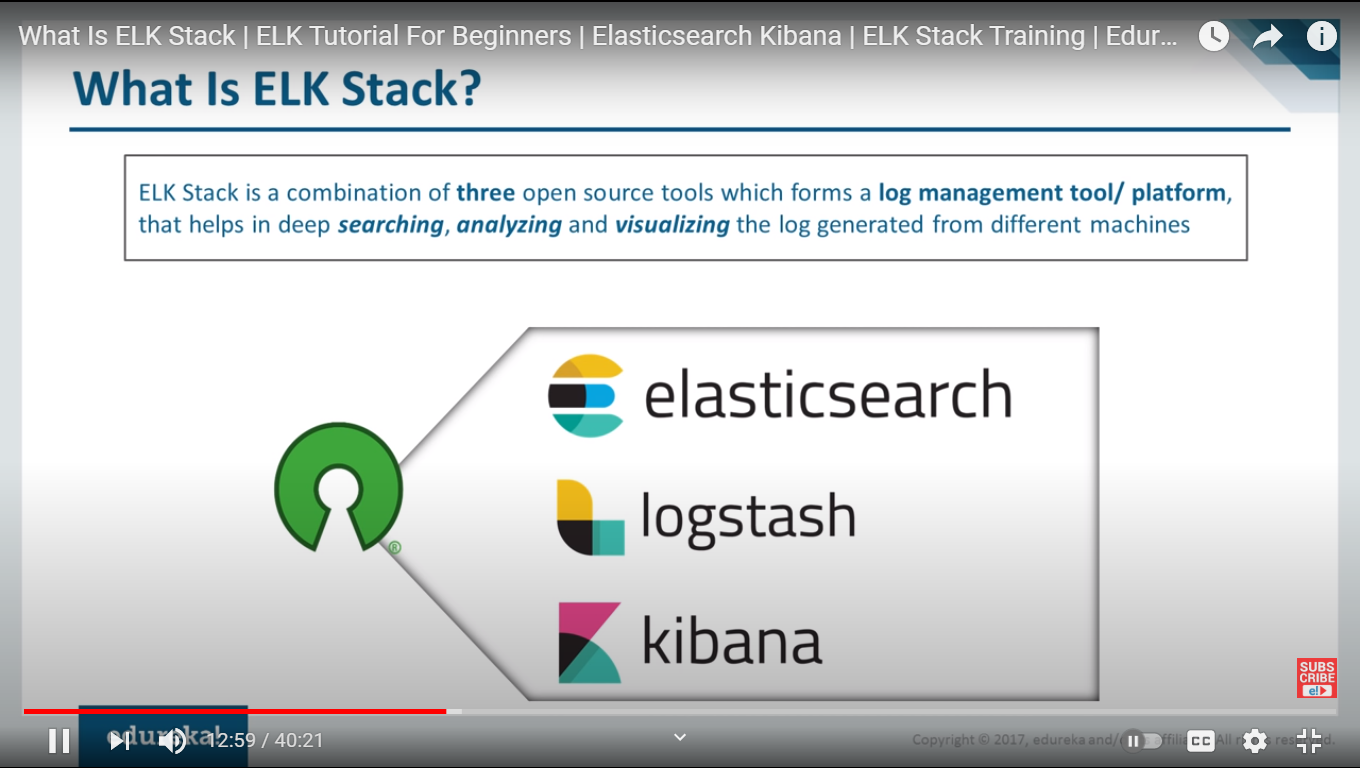
**Access via SSO** (Azure Active Directory Seamless Single Sign-On)

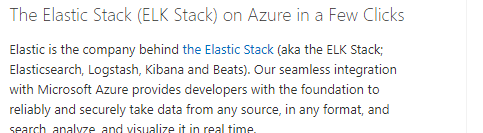
customers access Datadog from the Azure portal through a streamlined SSO experience and configure Datadog as a destination for logs and metrics from Azure services.



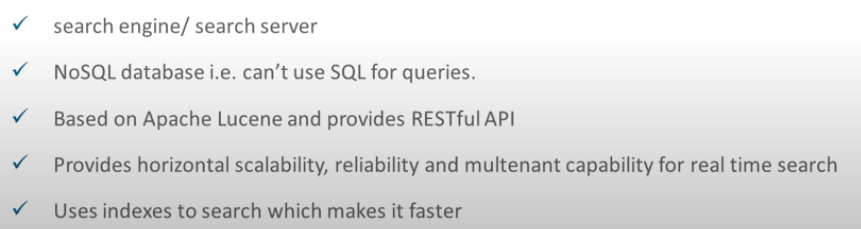
ELK Stack

The ELK stack is an acronym used to describe a stack that comprises of three popular projects: **Elasticsearch, Logstash, and Kibana**.

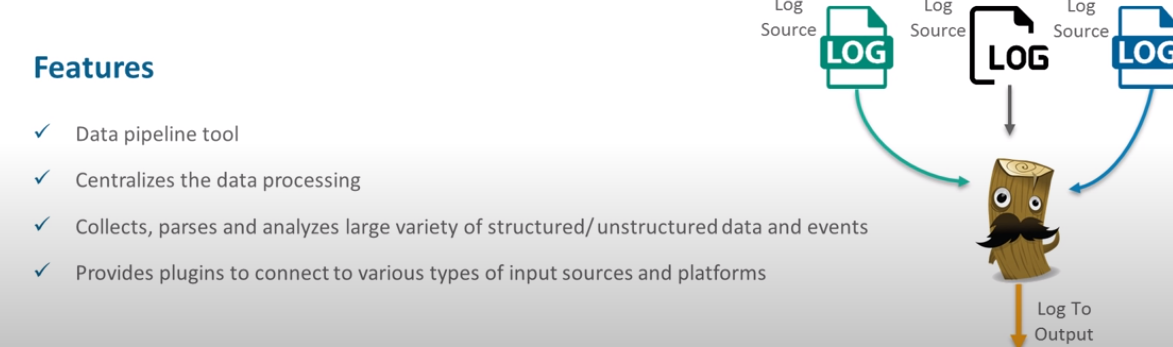




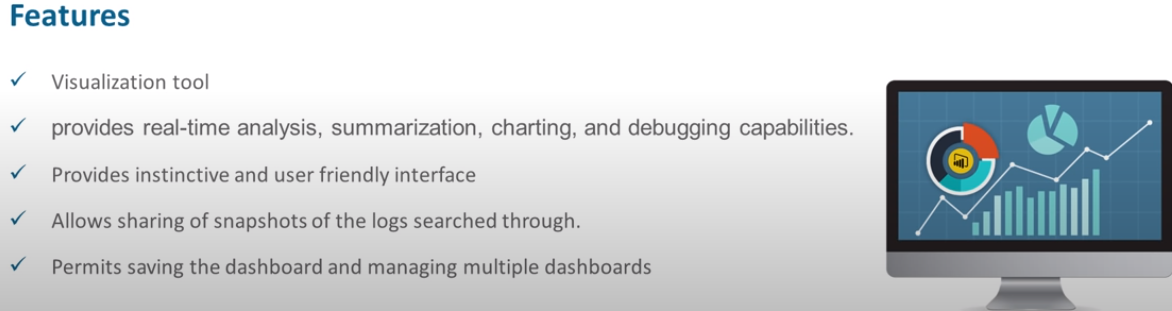
Elastic search

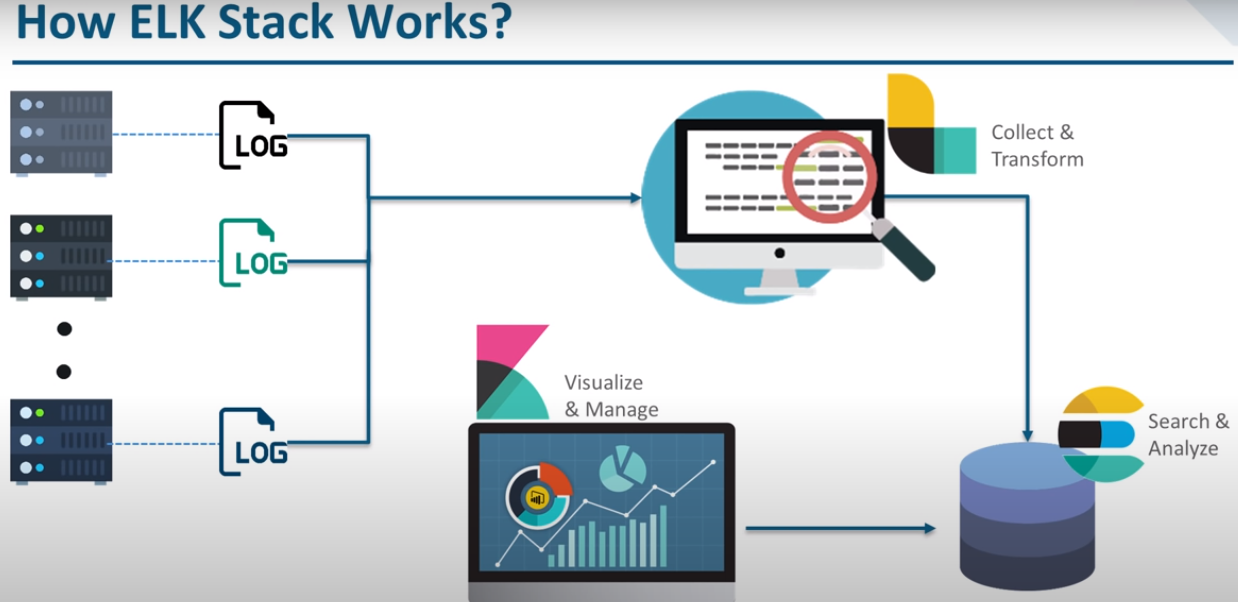


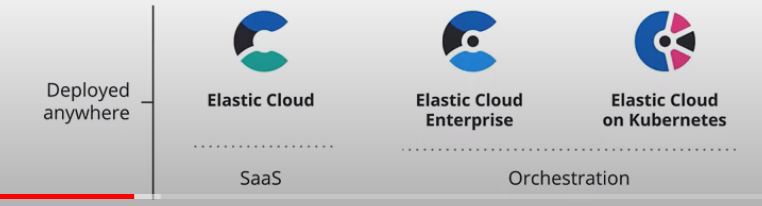
Log stash

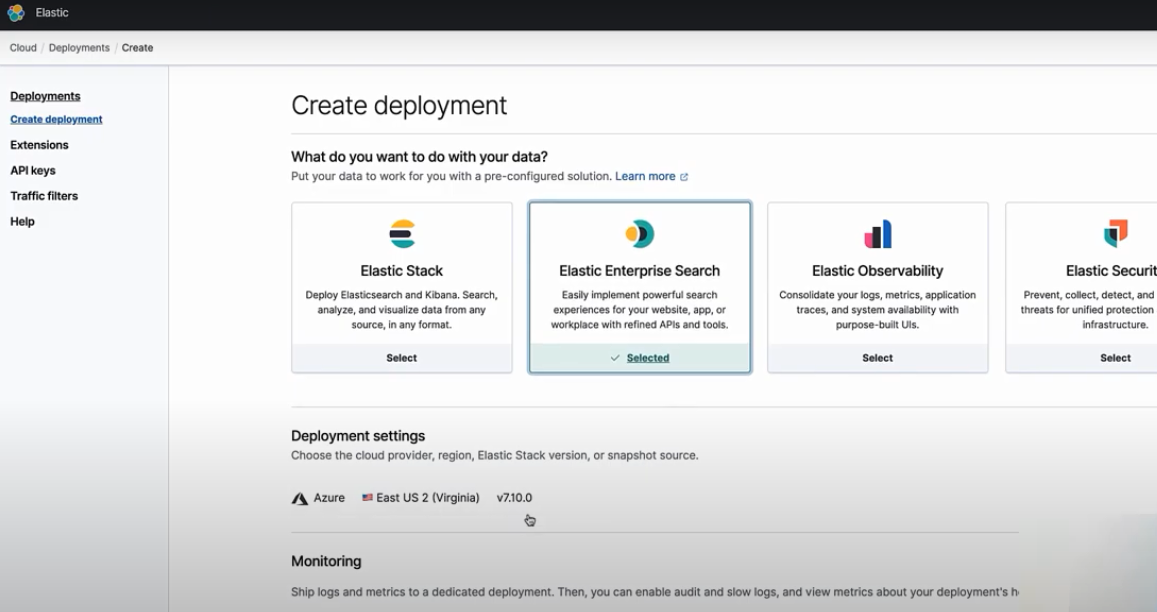
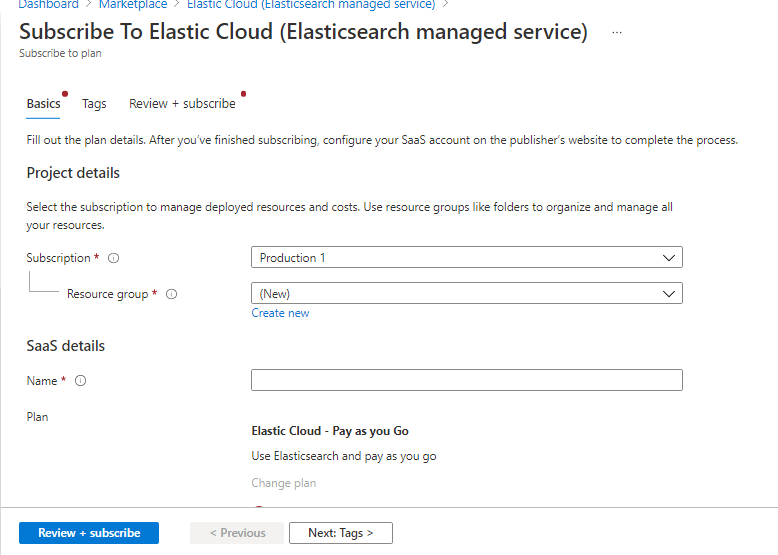


Kibana









# deploy elk stack on Azure

<https://www.youtube.com/watch?v=YE_02hCk1-c>

<https://github.com/chrisvugrinec/azure-kube-elk>

<https://www.youtube.com/watch?v=oxjPX1oSBDs>

## Micro srvices hosting deployment strategy :

<https://docs.openshift.com/container-platform/3.7/dev_guide/deployments/deployment_strategies.html>

What Are microservices Deployment Strategies?

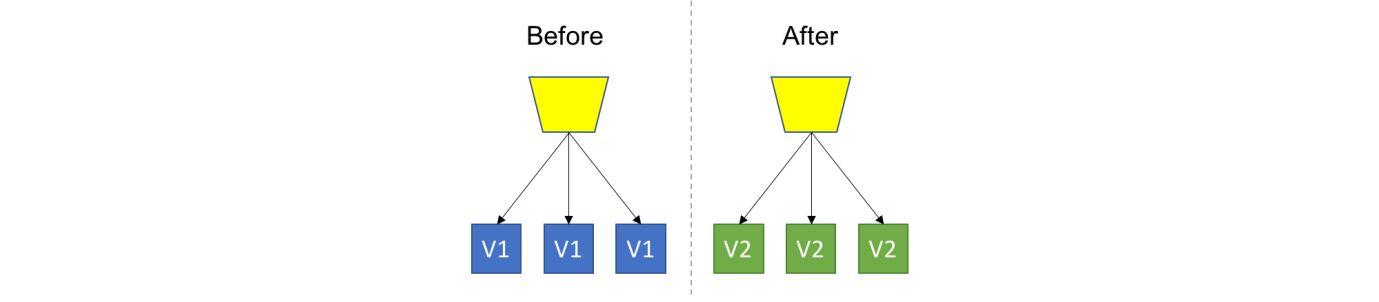
A deployment strategy is a way to change or upgrade an application. The aim is to make the change without downtime in a way that the user barely notices the improvements.

The most common strategy is to use a [blue-green deployment](https://docs.openshift.com/container-platform/3.7/dev_guide/deployments/advanced_deployment_strategies.html#advanced-deployment-strategies-blue-green-deployments). The new version (the blue version) is brought up for testing and evaluation, while the users still use the stable version (the green version). When ready, the users are switched to the blue version. If a problem arises, you can switch back to the green version.

A common alternative strategy is to use A/B versions that are both active at the same time and some users use one version, and some users use the other version. This can be used for experimenting with user interface changes and other features to get user feedback. It can also be used to verify proper operation in a production context where problems impact a limited number of users.

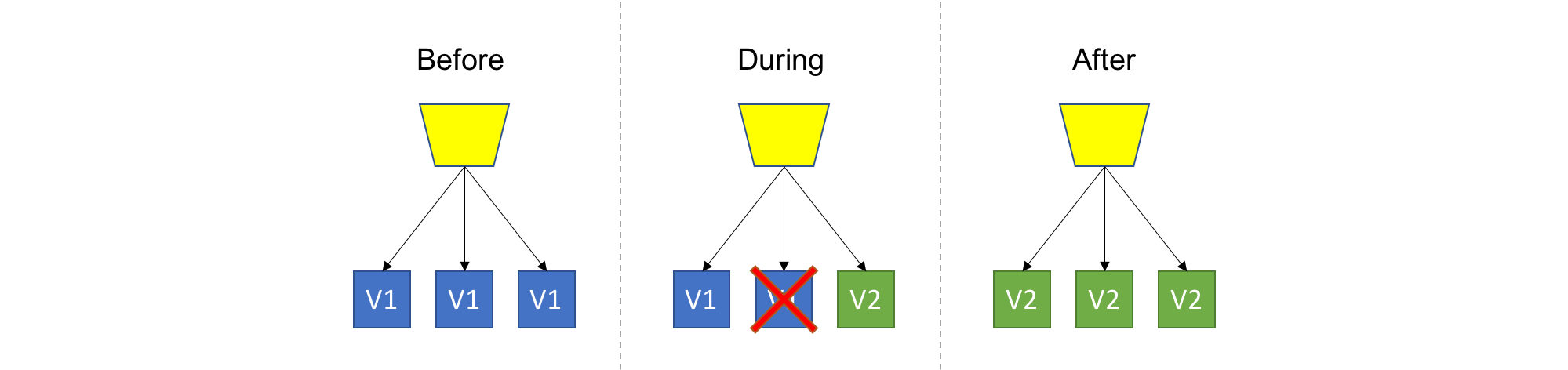
* Reckless Deployment - If you don’t care.

TL;DR: Destroy the old, provision the new, don’t care for consequences.



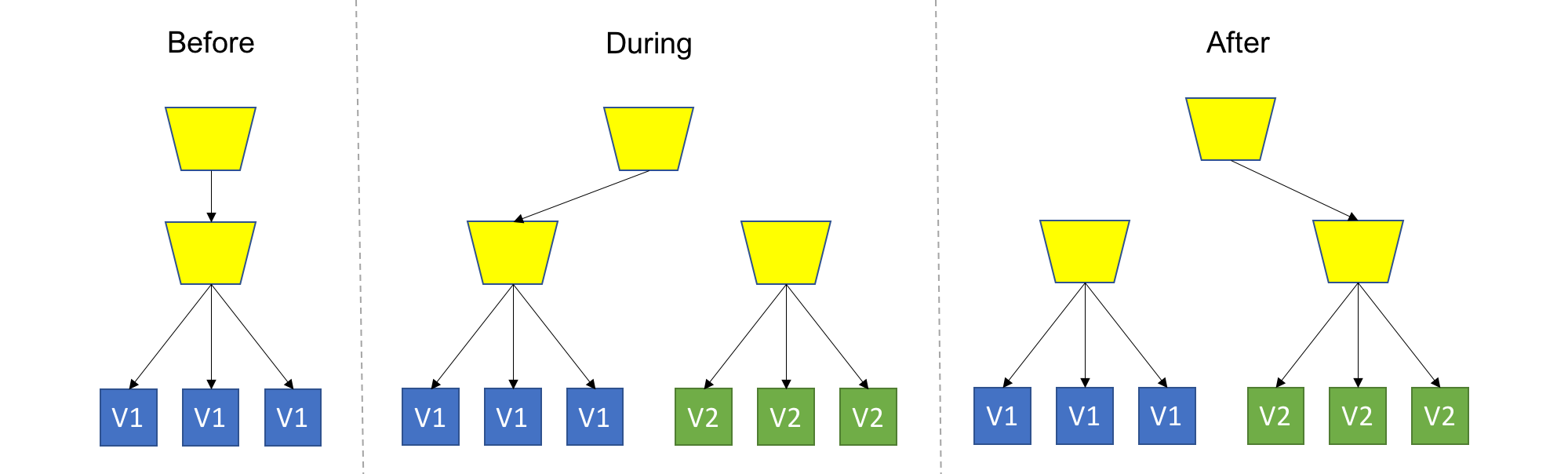
* Rolling Upgrade - Deploy gradually in order to allow time to roll back.

 Introduce new version of the code into the existing deployment, gradually ramping up while decommissioning old code.

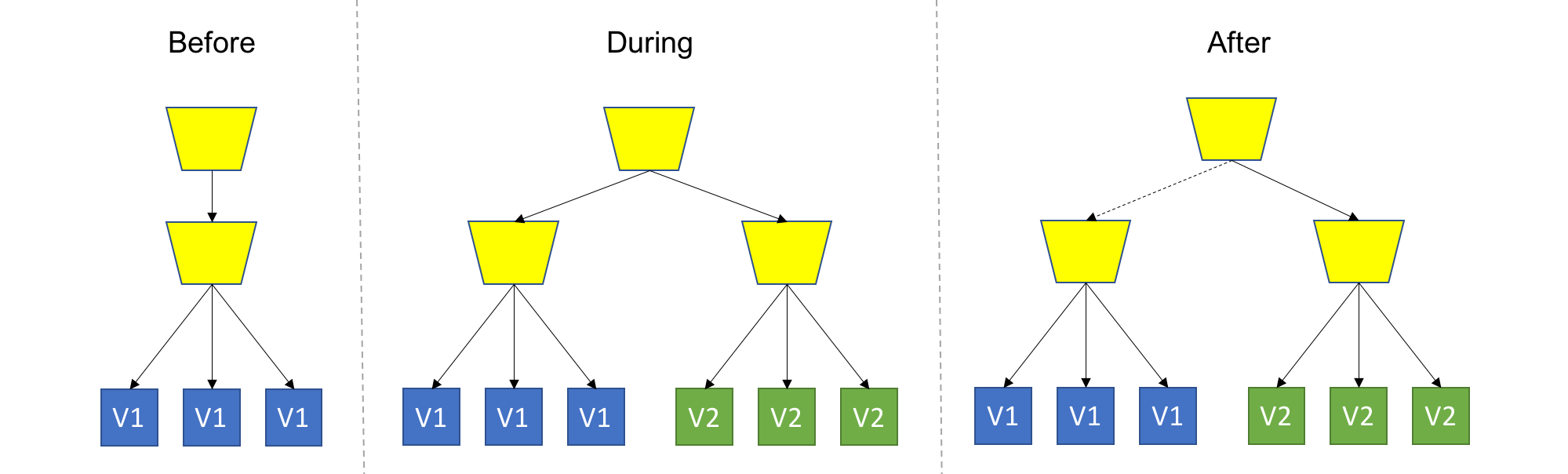


* Blue/Green Deployment - Encourages safety by protecting the production service from any altering operation, directing these operations to a side environment.

Spin up a new separate deployment for the new version, without affecting the current one. Test the new version, and once ready start routing users to the new version

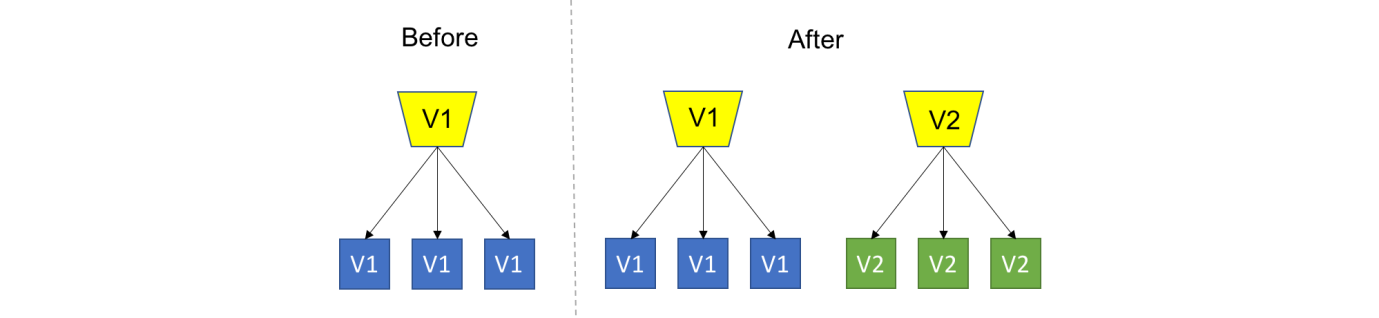


* Canary - Which exposes new versions to real users.
* a.k.a: Ringed Deployment, Feature toggling, Dark features, Silent launch, A/B testing, Testing in production.
* TL;DR: Deploy the new version into production alongside the old one, carefully controlling who gets to use the new version. Monitor and tune the experiment while gradually expanding it’s population



* Versioned Deployment - Which is a special purpose design that allows multiple versions to coexist forever.

TL;DR: Always keep all versions alive, while letting the user choose which version to use



<https://blog.itaysk.com/2017/11/20/deployment-strategies-defined>