

CI/CD pipeline for container-based workloads

07/05/2018 • 7 minutes to read • Contributors      all

In this article

[Relevant use cases](#)

[Architecture](#)

[Considerations](#)

[Deploy the scenario](#)

[Pricing](#)

[Related resources](#)

This example scenario is applicable to businesses that want to modernize application development by using containers and DevOps workflows. In this scenario, a Node.js web app is built and deployed by Jenkins into an Azure Container Registry and Azure Kubernetes Service. For a globally distributed database tier, Azure Cosmos DB is used. To monitor and troubleshoot application performance, Azure Monitor integrates with a Grafana instance and dashboard.

Example application scenarios include providing an automated development environment, validating new code commits, and pushing new deployments into staging or production environments. Traditionally, businesses had to manually build and compile applications and updates, and maintain a large, monolithic code base. With a modern approach to application development that uses continuous integration (CI) and continuous deployment (CD), you can more quickly build, test, and deploy services. This modern approach lets you release applications and updates to your customers faster, and respond to changing business demands in a more agile manner.

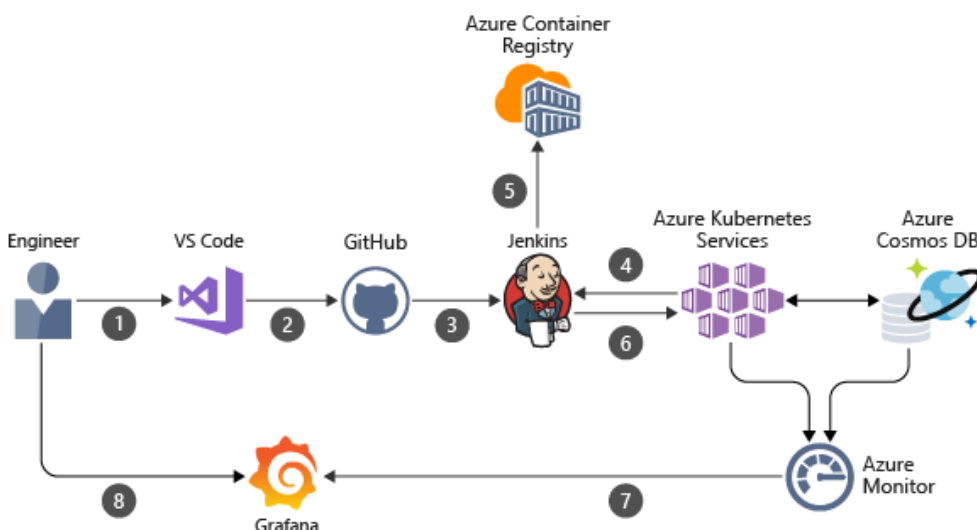
By using Azure services such as Azure Kubernetes Service, Container Registry, and Cosmos DB, companies can use the latest in application development techniques and tools to simplify the process of implementing high availability.

Relevant use cases

Other relevant use cases include:

- Modernizing application development practices to a microservice, container-based approach.
- Speeding up application development and deployment lifecycles.
- Automating deployments to test or acceptance environments for validation.

Architecture



This scenario covers a DevOps pipeline for a Node.js web application and database back end. The data flows through the scenario as follows:

1. A developer makes changes to the Node.js web application source code.
2. The code change is committed to a source control repository, such as GitHub.
3. To start the continuous integration (CI) process, a GitHub webhook triggers a Jenkins project build.
4. The Jenkins build job uses a dynamic build agent in Azure Kubernetes Service to perform a container build process.
5. A container image is created from the code in source control, and is then pushed to an Azure Container Registry.
6. Through continuous deployment (CD), Jenkins deploys this updated container image to the Kubernetes cluster.
7. The Node.js web application uses Cosmos DB as its back end. Both Cosmos DB and Azure Kubernetes Service report metrics to Azure Monitor.
8. A Grafana instance provides visual dashboards of the application performance based on the data from Azure Monitor.

Components

- [Jenkins](#) is an open-source automation server that can integrate with Azure services to enable continuous integration (CI) and continuous deployment (CD). In this scenario, Jenkins orchestrates the creation of new container images based on commits to source control, pushes those images to Azure Container Registry, then updates application instances in Azure Kubernetes Service.
- [Azure Linux Virtual Machines](#) is the IaaS platform used to run the Jenkins and Grafana instances.
- [Azure Container Registry](#) stores and manages container images that are used by the Azure Kubernetes Service cluster. Images are securely stored, and can be replicated to other regions by the Azure platform to speed up deployment times.
- [Azure Kubernetes Service](#) is a managed Kubernetes platform that lets you deploy and manage containerized applications without container orchestration expertise. As a hosted Kubernetes service, Azure handles critical tasks like health monitoring and maintenance for you.
- [Azure Cosmos DB](#) is a globally distributed, multi-model database that allows you to choose from various database and consistency models to suit your needs. With Cosmos DB, your data can be globally replicated, and there is no cluster management or replication components to deploy and configure.
- [Azure Monitor](#) helps you track performance, maintain security, and identify trends. Metrics obtained by Monitor can be used by other resources and tools, such as Grafana.
- [Grafana](#) is an open-source solution to query, visualize, alert, and understand metrics. A data source plugin for Azure Monitor allows Grafana to create visual dashboards to monitor the performance of your applications running in Azure Kubernetes Service and using Cosmos DB.

Alternatives

- [Azure Pipelines](#) help you implement a continuous integration (CI), test, and deployment (CD) pipeline for any app.
- [Kubernetes](#) can be run directly on Azure VMs instead of via a managed service if you would like more control over the cluster.
- [Service Fabric](#) is another alternate container orchestrator that can replace AKS.

Considerations

Availability

To monitor your application performance and report on issues, this scenario combines Azure Monitor with Grafana for visual dashboards. These tools let you monitor and troubleshoot performance issues that may require code updates, which can all then be deployed with the CI/CD pipeline.

As part of the Azure Kubernetes Service cluster, a load balancer distributes application traffic to one or more containers (pods) that run your application. This approach to running containerized applications in Kubernetes provides a highly available infrastructure for your customers.

Scalability

Azure Kubernetes Service lets you scale the number of cluster nodes to meet the demands of your applications. As your application increases, you can scale out the number of Kubernetes nodes that run your service.

Application data is stored in Azure Cosmos DB, a globally distributed, multi-model database that can scale globally. Cosmos DB abstracts the need to scale your infrastructure as with traditional database components, and you can choose to replicate your Cosmos DB globally to meet the demands of your customers.

For other scalability topics, see the [scalability checklist](#) available in the Azure Architecture Center.

Security

To minimize the attack footprint, this scenario does not expose the Jenkins VM instance over HTTP. For any management tasks that require you to interact with Jenkins, you create a secure remote connection using an SSH tunnel from your local machine. Only SSH public key authentication is allowed for the Jenkins and Grafana VM instances. Password-based logins are disabled. For more information, see [Run a Jenkins server on Azure](#).

For separation of credentials and permissions, this scenario uses a dedicated Azure Active Directory (AD) service principal. The credentials for this service principal are stored as a secure credential object in Jenkins so that they are not directly exposed and visible within scripts or the build pipeline.

For general guidance on designing secure solutions, see the [Azure Security Documentation](#).

Resiliency



This scenario uses Azure Kubernetes Service for your application. Built into Kubernetes are resiliency components that monitor and restart the containers (pods) if there is an issue. Combined with running multiple Kubernetes nodes, your application can tolerate a pod or node being unavailable.

For general guidance on designing resilient solutions, see [Designing reliable Azure applications](#).

Deploy the scenario


Prerequisites

- You must have an existing Azure account. If you don't have an Azure subscription, create a [free account](#) before you begin.
- You need an SSH public key pair. For steps on how to create a public key pair, see [Create and use an SSH key pair for Linux VMs](#).
- You need an Azure Active Directory (AD) service principal for the authentication of service and resources. If needed, you can create a service principal with [az ad sp create-for-rbac](#)

Azure CLI	 Copy	 Try It
<pre>az ad sp create-for-rbac --name myDevOpsScenario</pre>		

Make a note of the *appId* and *password* in the output from this command. You provide these values to the template when you deploy the scenario.

- Find the supported Kubernetes versions for your deployment region by running [az aks get-versions](#). The following command gets the [CLI default](#) version:

Azure CLI	 Copy	 Try It
<pre>az aks get-versions -l <region> --query "orchestrators[?default!=null].orchestratorVersion" -o tsv</pre>		

Walk-through

To deploy this scenario with an Azure Resource Manager template, perform the following steps.

1. Click the link below to deploy the solution.



2. Wait for the template deployment to open in the Azure portal, then complete the following steps:
 - Choose to **Create new** resource group, then provide a name such as *myAKSDevOpsScenario* in the text box.
 - Select a region from the **Location** drop-down box.
 - Enter your service principal app ID and password from the `az ad sp create-for-rbac` command.
 - Provide a username and secure password for the Jenkins instance and Grafana console.
 - Provide an SSH key to secure logins to the Linux VMs.
 - Enter the Kubernetes version from the `az aks get-versions` command.
 - Review the terms and conditions, then check **I agree to the terms and conditions stated above**.
 - Select the **Purchase** button.

It can take 15-20 minutes for the deployment to complete.

Pricing

To explore the cost of running this scenario, all of the services are pre-configured in the cost calculator. To see how the pricing would change for your particular use case, change the appropriate variables to match your expected traffic.

We have provided three sample cost profiles based on the number of container images to store and Kubernetes nodes to run your applications.

- **Small**: this pricing example correlates to 1000 container builds per month.
- **Medium**: this pricing example correlates to 100,000 container builds per month.
- **Large**: this pricing example correlates to 1,000,000 container builds per month.

Related resources

This scenario used Azure Container Registry and Azure Kubernetes Service to store and run a container-based application. Azure Container Instances can also be used to run container-based applications, without having to provision any orchestration components. For more information, see [Azure Container Instances overview](#).