**POWER SHELL SCRIPT**

**Sample script**

PowerShellCopy

# Variables for common values

$resourceGroup = "myResourceGroup"

$location = "westeurope"

$vmName = "myVM"

# Create user object

$cred = Get-Credential -Message "Enter a username and password for the virtual machine."

# Create a resource group

New-AzResourceGroup -Name $resourceGroup -Location $location

# Create a subnet configuration

$subnetConfig = New-AzVirtualNetworkSubnetConfig -Name mySubnet -AddressPrefix 192.168.1.0/24

# Create a virtual network

$vnet = New-AzVirtualNetwork -ResourceGroupName $resourceGroup -Location $location `

-Name MYvNET -AddressPrefix 192.168.0.0/16 -Subnet $subnetConfig

# Create a public IP address and specify a DNS name

$pip = New-AzPublicIpAddress -ResourceGroupName $resourceGroup -Location $location `

-Name "mypublicdns$(Get-Random)" -AllocationMethod Static -IdleTimeoutInMinutes 4

# Create an inbound network security group rule for port 3389

$nsgRuleRDP = New-AzNetworkSecurityRuleConfig -Name myNetworkSecurityGroupRuleRDP -Protocol Tcp `

-Direction Inbound -Priority 1000 -SourceAddressPrefix \* -SourcePortRange \* -DestinationAddressPrefix \* `

-DestinationPortRange 3389 -Access Allow

# Create a network security group

$nsg = New-AzNetworkSecurityGroup -ResourceGroupName $resourceGroup -Location $location `

-Name myNetworkSecurityGroup -SecurityRules $nsgRuleRDP

# Create a virtual network card and associate with public IP address and NSG

$nic = New-AzNetworkInterface -Name myNic -ResourceGroupName $resourceGroup -Location $location `

-SubnetId $vnet.Subnets[0].Id -PublicIpAddressId $pip.Id -NetworkSecurityGroupId $nsg.Id

# Create a virtual machine configuration

$vmConfig = New-AzVMConfig -VMName $vmName -VMSize Standard\_D1 | `

Set-AzVMOperatingSystem -Windows -ComputerName $vmName -Credential $cred | `

Set-AzVMSourceImage -PublisherName MicrosoftWindowsServer -Offer WindowsServer -Skus 2016-Datacenter -Version latest | `

Add-AzVMNetworkInterface -Id $nic.Id

# Create a virtual machine

New-AzVM -ResourceGroupName $resourceGroup -Location $location -VM $vmConfig

**Clean up deployment**

Run the following command to remove the resource group, VM, and all related resources.

PowerShellCopy

Remove-AzResourceGroup -Name myResourceGroup

**Script explanation**

This script uses the following commands to create the deployment. Each item in the table links to command specific documentation.

| **Table 1** | |
| --- | --- |
| **Command** | **Notes** |
| [New-AzResourceGroup](https://docs.microsoft.com/powershell/module/az.resources/new-azresourcegroup) | Creates a resource group in which all resources are stored. |
| [New-AzVirtualNetworkSubnetConfig](https://docs.microsoft.com/powershell/module/az.network/new-azvirtualnetworksubnetconfig) | Creates a subnet configuration. This configuration is used with the virtual network creation process. |
| [New-AzVirtualNetwork](https://docs.microsoft.com/powershell/module/az.network/new-azvirtualnetwork) | Creates a virtual network. |
| [New-AzPublicIpAddress](https://docs.microsoft.com/powershell/module/az.network/new-azpublicipaddress) | Creates a public IP address. |
| [New-AzNetworkSecurityRuleConfig](https://docs.microsoft.com/powershell/module/az.network/new-aznetworksecurityruleconfig) | Creates a network security group rule configuration. This configuration is used to create an NSG rule when the NSG is created. |
| [New-AzNetworkSecurityGroup](https://docs.microsoft.com/powershell/module/az.network/new-aznetworksecuritygroup) | Creates a network security group. |
| [Get-AzVirtualNetworkSubnetConfig](https://docs.microsoft.com/powershell/module/az.network/get-azvirtualnetworksubnetconfig) | Gets subnet information. This information is used when creating a network interface. |
| [New-AzNetworkInterface](https://docs.microsoft.com/powershell/module/az.network/new-aznetworkinterface) | Creates a network interface. |
| [New-AzVMConfig](https://docs.microsoft.com/powershell/module/az.compute/new-azvmconfig) | Creates a VM configuration. This configuration includes information such as VM name, operating system, and administrative credentials. The configuration is used during VM creation. |
| [New-AzVM](https://docs.microsoft.com/powershell/module/az.compute/new-azvm) | Create a virtual machine. |
| [Remove-AzResourceGroup](https://docs.microsoft.com/powershell/module/az.resources/remove-azresourcegroup) | Removes a resource group and all resources contained within. |

**Next steps**

For more information on the Azure PowerShell module, see [Azure PowerShell documentation](https://docs.microsoft.com/en-us/powershell/azure/overview).

Additional virtual machine PowerShell script samples can be found in the [Azure Windows VM documentation](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/powershell-samples?toc=/azure/virtual-machines/windows/toc.json).

**Related Articles**

**Is this page helpful?**

YesNo

**Top of Form**

**JSON TEMPLATE**

Dismiss

### Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage

## [azure-quickstart-templates](https://github.com/Azure/azure-quickstart-templates)/[101-vm-simple-windows](https://github.com/Azure/azure-quickstart-templates/tree/master/101-vm-simple-windows)/****azuredeploy.json****

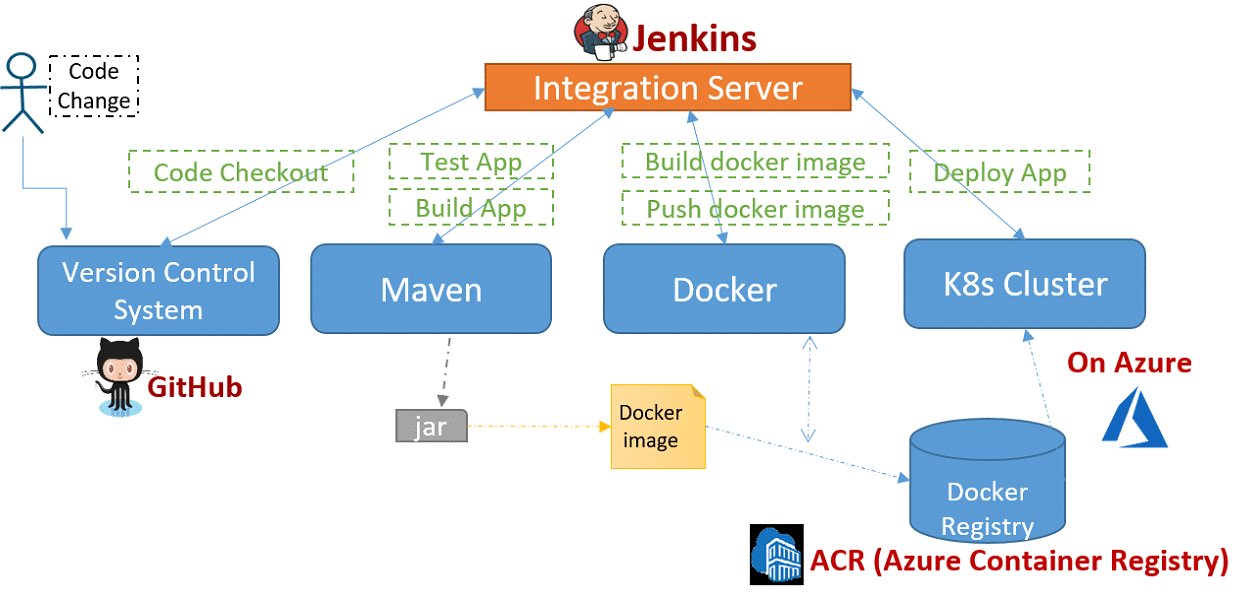
[Find file](https://github.com/Azure/azure-quickstart-templates/find/master) Copy path

|  |  |
| --- | --- |
|  | { |
|  | "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#", |
|  | "contentVersion": "1.0.0.0", |
|  | "parameters": { |
|  | "adminUsername": { |
|  | "type": "string", |
|  | "metadata": { |
|  | "description": "Username for the Virtual Machine." |
|  | } |
|  | }, |
|  | "adminPassword": { |
|  | "type": "securestring", |
|  | "metadata": { |
|  | "description": "Password for the Virtual Machine." |
|  | } |
|  | }, |
|  | "dnsLabelPrefix": { |
|  | "type": "string", |
|  | "metadata": { |
|  | "description": "Unique DNS Name for the Public IP used to access the Virtual Machine." |
|  | } |
|  | }, |
|  | "windowsOSVersion": { |
|  | "type": "string", |
|  | "defaultValue": "2016-Datacenter", |
|  | "allowedValues": [ |
|  | "2008-R2-SP1", |
|  | "2012-Datacenter", |
|  | "2012-R2-Datacenter", |
|  | "2016-Nano-Server", |
|  | "2016-Datacenter-with-Containers", |
|  | "2016-Datacenter", |
|  | "2019-Datacenter" |
|  | ], |
|  | "metadata": { |
|  | "description": "The Windows version for the VM. This will pick a fully patched image of this given Windows version." |
|  | } |
|  | }, |
|  | "vmSize": { |
|  | "type": "string", |
|  | "defaultValue": "Standard\_D2\_v3", |
|  | "metadata": { |
|  | "description": "Size of the virtual machine." |
|  | } |
|  | }, |
|  | "location": { |
|  | "type": "string", |
|  | "defaultValue": "[resourceGroup().location]", |
|  | "metadata": { |
|  | "description": "Location for all resources." |
|  | } |
|  | } |
|  | }, |
|  | "variables": { |
|  | "storageAccountName": "[concat(uniquestring(resourceGroup().id), 'sawinvm')]", |
|  | "nicName": "myVMNic", |
|  | "addressPrefix": "10.0.0.0/16", |
|  | "subnetName": "Subnet", |
|  | "subnetPrefix": "10.0.0.0/24", |
|  | "publicIPAddressName": "myPublicIP", |
|  | "vmName": "SimpleWinVM", |
|  | "virtualNetworkName": "MyVNET", |
|  | "subnetRef": "[resourceId('Microsoft.Network/virtualNetworks/subnets', variables('virtualNetworkName'), variables('subnetName'))]", |
|  | "networkSecurityGroupName": "default-NSG" |
|  | }, |
|  | "resources": [ |
|  | { |
|  | "type": "Microsoft.Storage/storageAccounts", |
|  | "apiVersion": "2018-11-01", |
|  | "name": "[variables('storageAccountName')]", |
|  | "location": "[parameters('location')]", |
|  | "sku": { |
|  | "name": "Standard\_LRS" |
|  | }, |
|  | "kind": "Storage", |
|  | "properties": {} |
|  | }, |
|  | { |
|  | "type": "Microsoft.Network/publicIPAddresses", |
|  | "apiVersion": "2018-11-01", |
|  | "name": "[variables('publicIPAddressName')]", |
|  | "location": "[parameters('location')]", |
|  | "properties": { |
|  | "publicIPAllocationMethod": "Dynamic", |
|  | "dnsSettings": { |
|  | "domainNameLabel": "[parameters('dnsLabelPrefix')]" |
|  | } |
|  | } |
|  | }, |
|  | { |
|  | "comments": "Default Network Security Group for template", |
|  | "type": "Microsoft.Network/networkSecurityGroups", |
|  | "apiVersion": "2019-08-01", |
|  | "name": "[variables('networkSecurityGroupName')]", |
|  | "location": "[parameters('location')]", |
|  | "properties": { |
|  | "securityRules": [ |
|  | { |
|  | "name": "default-allow-3389", |
|  | "properties": { |
|  | "priority": 1000, |
|  | "access": "Allow", |
|  | "direction": "Inbound", |
|  | "destinationPortRange": "3389", |
|  | "protocol": "Tcp", |
|  | "sourcePortRange": "\*", |
|  | "sourceAddressPrefix": "\*", |
|  | "destinationAddressPrefix": "\*" |
|  | } |
|  | } |
|  | ] |
|  | } |
|  | }, |
|  | { |
|  | "type": "Microsoft.Network/virtualNetworks", |
|  | "apiVersion": "2018-11-01", |
|  | "name": "[variables('virtualNetworkName')]", |
|  | "location": "[parameters('location')]", |
|  | "dependsOn": [ |
|  | "[resourceId('Microsoft.Network/networkSecurityGroups', variables('networkSecurityGroupName'))]" |
|  | ], |
|  | "properties": { |
|  | "addressSpace": { |
|  | "addressPrefixes": [ |
|  | "[variables('addressPrefix')]" |
|  | ] |
|  | }, |
|  | "subnets": [ |
|  | { |
|  | "name": "[variables('subnetName')]", |
|  | "properties": { |
|  | "addressPrefix": "[variables('subnetPrefix')]", |
|  | "networkSecurityGroup": { |
|  | "id": "[resourceId('Microsoft.Network/networkSecurityGroups', variables('networkSecurityGroupName'))]" |
|  | } |
|  | } |
|  | } |
|  | ] |
|  | } |
|  | }, |
|  | { |
|  | "type": "Microsoft.Network/networkInterfaces", |
|  | "apiVersion": "2018-11-01", |
|  | "name": "[variables('nicName')]", |
|  | "location": "[parameters('location')]", |
|  | "dependsOn": [ |
|  | "[resourceId('Microsoft.Network/publicIPAddresses/', variables('publicIPAddressName'))]", |
|  | "[resourceId('Microsoft.Network/virtualNetworks/', variables('virtualNetworkName'))]" |
|  | ], |
|  | "properties": { |
|  | "ipConfigurations": [ |
|  | { |
|  | "name": "ipconfig1", |
|  | "properties": { |
|  | "privateIPAllocationMethod": "Dynamic", |
|  | "publicIPAddress": { |
|  | "id": "[resourceId('Microsoft.Network/publicIPAddresses',variables('publicIPAddressName'))]" |
|  | }, |
|  | "subnet": { |
|  | "id": "[variables('subnetRef')]" |
|  | } |
|  | } |
|  | } |
|  | ] |
|  | } |
|  | }, |
|  | { |
|  | "type": "Microsoft.Compute/virtualMachines", |
|  | "apiVersion": "2018-10-01", |
|  | "name": "[variables('vmName')]", |
|  | "location": "[parameters('location')]", |
|  | "dependsOn": [ |
|  | "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]", |
|  | "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]" |
|  | ], |
|  | "properties": { |
|  | "hardwareProfile": { |
|  | "vmSize": "[parameters('vmSize')]" |
|  | }, |
|  | "osProfile": { |
|  | "computerName": "[variables('vmName')]", |
|  | "adminUsername": "[parameters('adminUsername')]", |
|  | "adminPassword": "[parameters('adminPassword')]" |
|  | }, |
|  | "storageProfile": { |
|  | "imageReference": { |
|  | "publisher": "MicrosoftWindowsServer", |
|  | "offer": "WindowsServer", |
|  | "sku": "[parameters('windowsOSVersion')]", |
|  | "version": "latest" |
|  | }, |
|  | "osDisk": { |
|  | "createOption": "FromImage" |
|  | }, |
|  | "dataDisks": [ |
|  | { |
|  | "diskSizeGB": 1023, |
|  | "lun": 0, |
|  | "createOption": "Empty" |
|  | } |
|  | ] |
|  | }, |
|  | "networkProfile": { |
|  | "networkInterfaces": [ |
|  | { |
|  | "id": "[resourceId('Microsoft.Network/networkInterfaces',variables('nicName'))]" |
|  | } |
|  | ] |
|  | }, |
|  | "diagnosticsProfile": { |
|  | "bootDiagnostics": { |
|  | "enabled": true, |
|  | "storageUri": "[reference(resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))).primaryEndpoints.blob]" |
|  | } |
|  | } |
|  | } |
|  | } |
|  | ], |
|  | "outputs": { |
|  | "hostname": { |
|  | "type": "string", |
|  | "value": "[reference(variables('publicIPAddressName')).dnsSettings.fqdn]" |
|  | } |
|  | } |
|  | }  POLICY DETAILS FOR ROLE / USER/ GROUP ACCESS IN IAM |

**PYTHON JENKIN CI/CD SETUP**

# CI/CD Series - Part 1: How to build a CI/CD pipeline using Jenkins, Docker & Kubernetes? (Step-by-Step Guide)

CI/CD has become a very important term in modern software development process, in particular microservice development. It lays out a set of principles that enable development teams to deliver value faster, more reliably and transparently. Recently, many different tools that enable CI/CD have emerged in the market, carrying different features and aspects. As a result, this CI/CD Blog-Series is introduced, such that in each Blogpost a different tool will be used.



**Intro**

CI/CD has become a very important term in modern software development process, in particular microservice development.  It lays out a set of principles that enable development teams to deliver value faster, more reliably and transparently. The continuous integration part refers to building applications continuously, including all defined sets of tests (for example, including smoke, integration or business-logic tests). The continuous deployment part enables deploying tested and packaged applications on an environment for release continuously. If applied correctly, a working DevOps model can be achieved.

Recently, many different tools that enable CI/CD have emerged in the market, carrying different features and aspects. As a result, this CI/CD Blog-Series is introduced, such that in each Blogpost a different tool will be used.

This Blogpost is the first in the CI/CD Blog-Series and within it we will set up a CI/CD pipeline for a containerized (Docker) application on Kubernetes using *jenkins* as a tool.

*Please note that in the Blog, we will use Azure as a provider to create our Kubernetes Cluster and a private Docker registry. However, you can use any Kubernetes Cluster/Docker registry that you already have on any provider or locally on your machine to complete the steps here. You don’t need to touch the application code or the jenkinsfile itself!*

**Objective**

At the end of the post, you should be able to build a CI/CD pipeline for a sample spring-boot java application using Github as a version control system, Maven as a build tool, jenkins as an integration server, Docker as a container technology and Kubernetes on Azure as a platform. The end goal is to automate the following process:

* Checkout code.
* Compile code.
* Run test cases.
* Build docker images.
* Push images to docker registry.
* Pull new images from registry.
* Deploy the app on Kubernetes.

**Prerequisites**

In order to follow the steps in this guide, you will need the following:

* [GitHub Account](https://github.com): You need a GitHub account to fork/clone the ToDo app repository.
* [Azure Microsoft Account](https://azure.microsoft.com/en-us/): You need an Azure Microsoft account in order to use Azure.
* [Azure CLI](https://docs.microsoft.com/cli/azure/install-azure-cli?view=azure-cli-latest): You need to install the current version of Azure CLI  in order to create a Kubernetes cluster on Azure.
* [kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl/): You need to install the kubectl command-line interface in order to run commands against the Kubernetes cluster.
* [Helm](https://github.com/kubernetes/helm/blob/master/docs/install.md): You need to install Helm and Tiller in order to help you install jenkins.

**Get the Sample Application from Github**

The sample application code is hosted on [Github](https://github.com/MirnaAlaisami/CICD). You can clone/fork the repo to your own GitHub account.

The application consists of 3 main components:

* UI (todoui)
* Main app (todobackend)
* DB (postgresdb)

*todoui* and *todobackend* are Spring Boot apps with the following external configuration possibilities:

todoui:

**backend.host=${BACKEND\_HOST:todobackend}  
backend.port=${BACKEND\_PORT:8090}  
backend.url=http://${backend.host}:${backend.port}**

todobackend:

**spring.datasource.url= jdbc:postgresql://${POSTGRES\_HOST:postgresdb}:5432/mydb**

The application directory contains the *Dockerfiles* that are used to containerize the app components, the *yaml files* that are used to deploy the app on Kubernetes and the *jenkinsfile* that contains the CI/CD pipeline code which will be executed by jenkins.

**Create and Configure a Managed Kubernetes Cluster with a Container Registry Instance on Azure**

In order to create a managed K8s cluster on Azure, execute the following steps using the installed Azure CLI:

* Sign in with your account credentials in the browser.

**az login**

* Create a resource group. The following example creates a resource group named CICDResourceGroup in the westeurope location.

**az group create ––name CICDResourceGroup ––location westeurope**

* Create an Azure Container Registry (ACR) instance and give it a name. In the following example acrCICD is used.

**az acr create ––resource-group CICDResourceGroup ––name acrCICD ––sku Basic**

* Get and note the loginServer name, which will be used as the Docker Registry URL later on.

**az acr list ––resource-group CICDResourceGroup ––query “[].{acrLoginServer:loginServer}” ––output table**

* Get your Container Registry credentials and note the Docker registry **username** and **password**, which will be needed later on.

**az acr credential show ––name acrCICD**

* Create a service principal in order to allow an AKS cluster to interact with other Azure resources. Record the resulted values of (**appId**, **password**)

**az ad sp create-for-rbac ––skip-assignment**

This is an example output:

{  
“appId”: “00000000-5ee0-9900-faee-0000b9ee0000”,  
“displayName”: “azure-cli-2019-06-28-10-58-10”,  
“name”: “http://azure-cli-2019-06-28-10-58-10”,  
“password”: “00000000-ed00-5500-0000-aaaa0000dddd”,  
“tenant”: “a33000000-5555-0000e-0000-66ffffddd00”  
}

* Get the ACR resource ID <**acrId**>.

**az acr show ––name acrCICD ––resource-group CICDResourceGroup ––query “id” ––output tsv**

This is an example output:

/subscriptions/11110000-9999-eeee-aaaa-0d0d0333000d/resourceGroups/CICDResourceGroup/providers/Microsoft.ContainerRegistry/registries/acrCICD

* Create a role assignment in order to grant the correct access for the AKS cluster to use images stored in ACR, using the resulted <**appId**> and <**acrId**> in previuos steps.

**az role assignment create ––assignee 00000000-5ee0-9900-faee-0000b9ee0000 ––scope /subscriptions/11110000-9999-eeee-aaaa-0d0d0333000d/resourceGroups/CICDResourceGroup/providers/Microsoft.ContainerRegistry/registries/acrCICD ––role AcrPush**

* Create the Kubernetes cluster, give it a name and grant it access to the ACR, using the <**appId**> and <**password**> created before. You can choose the desired number of nodes in your created cluster (for this guide, one node is enough). Note: this step can take some time to complete.

**az aks create ––name CICDAKSCluster ––resource-group CICDResourceGroup ––node-count 1 ––generate-ssh-keys ––service-principal 00000000-5ee0-9900-faee-0000b9ee0000 ––client-secret 00000000-ed00-5500-0000-aaaa0000dddd**

* Configure kubectl to connect to your Kubernetes cluster.

**az aks get-credentials ––name CICDAKSCluster ––resource-group CICDResourceGroup**

**Deploy Jenkins on Kubernetes**

In order to deploy Jenkins with Helm, execute the following steps:

* Deploy jenkins with Helm using default configuration.

**helm install ––name cicd stable/jenkins**

After executing the previous command, you will get a “Notes” part on your screen that guides you to get the admin-password and access jenkins.

If you want to customize the configuration before deploying jenkins, you can clone the [Jenkins Helm repository](https://github.com/helm/charts) and update the file stable/jenkins/values.yaml, or use helm’s “-set Parameter” for overwriting values.

**Create a GitHub Webhook**

In order to integrate the GitHub repository into Jenkins, a webhook can be used to run the Jenkins build whenever a code commit is made in GitHub. To create the GitHub webhook, execute the following steps:

* In Github, move to your repository and select **“Settings”**.
* On the left-hand side, select **“Webhooks”**.
* Click on the **“Add webhook”** button.
* In the **“Payload URL”,** enter http://<publicIp:8080>/github-webhook/, replacing the <publicIp> with the IP address of the Jenkins server.
* Leave the options with the default selection and click on the **“Add webhook”**.

**Configure Jenkins**

Install the plugins in Jenkins by executing the following steps within the Jenkins dashboard (some plug-ins are already installed) :

* Select **Manage Jenkins** > **Manage Plugins** > **Available**.
* Search for and install the **Github, Kubernetes Cli, Blue Ocean** plug-in.

In order to enable Jenkins to launch pods for running jobs, you have to configure the service account credentials:

* Using jenkins dashboard, navigate to **Manage Jenkins** > **Configure System**> **Cloud** > **Credentials**, then select **“Add”.**
* Choose the type **“Kubernetes Service Account“**.

Using the Docker registry **username** and **password** that you obtained in the previous steps, add your Azure Docker registry credentials as the type **“Username with password“**:

* Using jenkins dashboard, navigate to **Credentials** > **System** > **Global Credentials**> **Add** **Credentials**.
* Choose the type **“Username with password“**.

Now it’s the time to configure the Jenkins slaves which will execute the jobs within the different stages:

* Using jenkins dashboard, navigate to **Manage Jenkins** > **Configure System**> **Cloud** > **Kubernetes Pod Template.**
* Inside the default pod, create the necessary **Container Templates** for *jnlp, Docker, Kubectl* and *Maven* (by giving suitables names and docker images for them).
* Click on **“Save“**.

**Create and Understand the Pipeline**

Before we start with that, let’s understand some basics. Jenkins is composed of two main components, the master that schedules the builds and dispatches the jobs, and the slave(s) that executes the build jobs dispatched from master.

When Jenkins master schedules a new build, it creates a Jenkins slave pod, in which different containers will be created depending on the stages that you define in your pipeline, such that each stage will be executed using a specified container and once the jobs in a stage are done, the container is shutdown. This interaction between jenkins and the kubernetes cluster, in order to start and stop slave/agent containers, is done using the Jenkins Kubernetes plugin, using a kubernetes service account.

Using **Blue Ocean**, the new UI for jenkins, you can easily create a new project and connect it to the Github repository, following these steps:

* Create a **new pipeline** and select **GitHub** as your code store.
* Enter a **Personal Access Token** from GitHub (which can be generated for the first time using GitHub **New Personal Access Token** page), so that Jenkins can access your private repositories.
* Choose your **Github repository** that contains the code.
* Give your pipeline a **name** and **save** the changes.

Let us now have a deep look into the content of the *jenkinsfile*! But before we go more into the details, note that all the variables that are written in capital letters are environment variables which take their values by configuring them globally using Jenkins UI.

*As mentioned at the beginning of the Blogpost, in this way, you can build this pipeline using any private or public Docker registry and a K8s Cluster on any provider that you choose. All you have to do is to configure these variables correctly giving them the suitable values, thus you don’t need to touch the jenkinsfile at all!*

In order to configure any environment variable using jenkins UI, follow these steps:

* Navigate to **Manage Jenkins** > **Configure System**> **Global properties**> **Environment variables**
* Give the **Name** and the **Value** of the variable.
* Click on **Save**.

Now, let’s have a look on the different parts of the jenkinsfile:

* Define some variables that will be used within the stages of your jenkinsfile. Here we are defining the names of the apps, the image tags, the container names and the dockerfile names. Note that the value of the environment variable “REPOSITORY” should be configured in the Jenkins UI as mentioned before, so that you can use your chosen Docker registry later on. The “BUILD\_NUMBER” will take its proper value automatically. Using the values of the Docker registry created in this example, the Docker environment variables will have the following values:
  + REPOSITORY -> acrcicd.azurecr.io

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //Define all variables  def app1\_name = 'todobackend'  def app2\_name = 'todoui'  def app1\_image\_tag = "${env.REPOSITORY}/${app1\_name}:v${env.BUILD\_NUMBER}"  def app2\_image\_tag = "${env.REPOSITORY}/${app2\_name}:v${env.BUILD\_NUMBER}"  def app1\_dockerfile\_name = 'Dockerfile-todobackend'  def app2\_dockerfile\_name = 'Dockerfile-todoui'  def app1\_container\_name = 'todobackend'  def app2\_container\_name = 'todoui' |

* Clone updates from the version control system, in our case here “Github”.



|  |  |
| --- | --- |
| 1  2  3  4 | //Stage 1: Checkout Code from Git      stage('Application Code Checkout from Git') {          checkout scm      } |

* Test the *todobackend* app with H2  in-memory mode using *Maven*. In order to execute the steps within this stage, we use the “maven” container, that we have already defined as a container Template in the “Configure System” section.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //Stage 2: Test Code with Maven/built-in Memory      stage('Test with Maven/H2') {          container('maven'){              dir ("./${app1\_name}") {                    sh ("mvn test -Dspring.profiles.active=dev")             }          }      } |

* Test the *todobackend* app with Postgres DB using *Maven*. In order to execute the steps within this stage, we first deploy a test Postgres DB as a pod in our K8s Cluster using the “kubectl” container. Then we test the connection of the app to it using “maven” as a container. Note that the values of the K8s environment variables should be configured in the Jenkins UI, so that you can connect to your K8s Cluster properly.  
  Using the values of the cluster created in this example,  the K8s environment variables will have the following values:
  + K8s\_CREDENTIALS\_ID -> (You should get this value from the credentials that you created in the “Configure Jenkins” section)
  + K8s\_SERVER\_URL -> https://cicdaksclu-cicdresourcegrou-000000-00000000.hcp.westeurope.azmk8s.io:443
  + K8s\_CONTEXT\_NAME -> CICDAKSCluster
  + CLUSTER\_NAME -> CICDAKSCluster

For the other DB Environment variables, consider the following values:

* + DB\_NAME -> mydb
  + DB\_USERNAME -> user
  + DB\_PASSWORD -> password



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | //Stage 3: Test Code with Maven/DB      stage('Test with Maven/PSQL') {          container('kubectl'){              withKubeConfig([credentialsId: env.K8s\_CREDENTIALS\_ID,              serverUrl: env.K8s\_SERVER\_URL,              contextName: env.K8s\_CONTEXT\_NAME,              clusterName: env.K8s\_CLUSTER\_NAME]){                    sh("kubectl apply -f postgres\_test.yml")              }            }            container('maven'){              dir ("./${app1\_name}") {                  sh ("mvn test -Dspring.profiles.active=prod -Dspring.datasource.url=jdbc:postgresql://${env.PSQL\_TEST}/${env.DB\_NAME} -Dspring.datasource.username=${env.DB\_USERNAME} -Dspring.datasource.password=${env.DB\_PASSWORD}")              }          }      } |

* Build the application jars. After the tests have passed, it is the time to use “Maven” to build the code, compile it, package it and put the jar files in the appropriate folders.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | //Stage 4: Build with mvn      stage('Build with Maven') {          container('maven'){              dir ("./${app1\_name}") {                    sh ("mvn -B -DskipTests clean package")              }              dir ("./${app2\_name}") {                    sh ("mvn -B -DskipTests clean package")              }          }      } |

* Build docker images. In this stage, we build the Docker images of the *todobackend* and *todoui* apps using the created jar files in the previous stage.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | //Stage 5: Build Docker Image      stage('Build Docker Image') {          container('docker'){              sh("docker build -f ${app1\_dockerfile\_name} -t ${app1\_image\_tag} .")              sh("docker build -f ${app2\_dockerfile\_name} -t ${app2\_image\_tag} .")          }      } |

* Login to the Docker Registry and push the Docker images created before to the registry. Note that the values of the Docker environment variables should be configured in the Jenkins UI, so that you can connect to your Docker Registry properly.  
  Using the values of the Docker registry created in this example, the Docker environment variables will have the following values:
  + DOCKER\_CREDENTIALS\_ID -> (You should get this value from the credentials that you created in the “Configure Jenkins” section)
  + DOCEKR\_REGISTRY -> http://acrcicd.azurecr.io

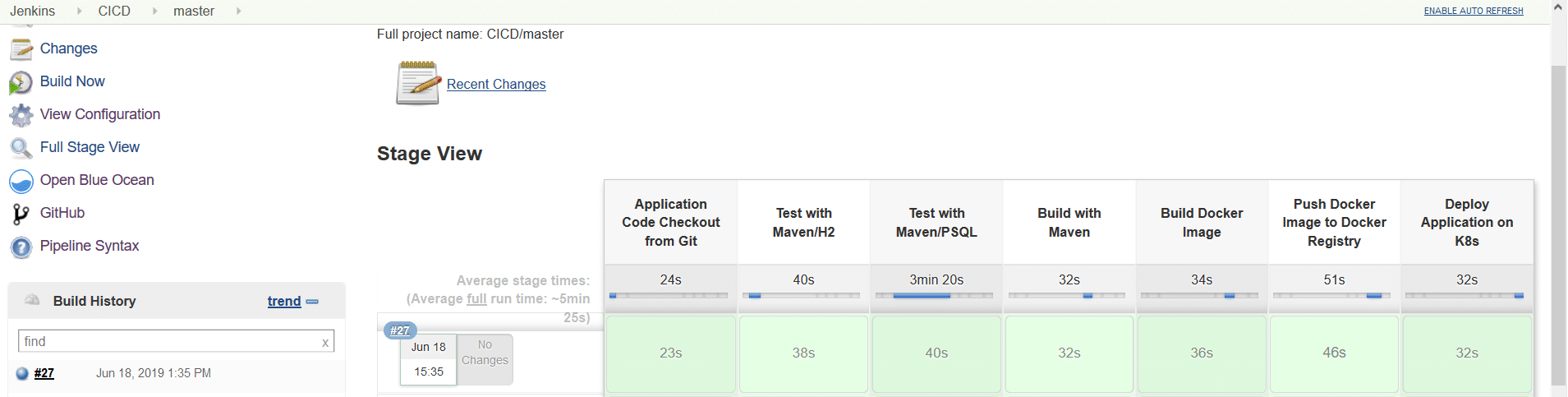


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | //Stage 6: Push the Image to a Docker Registry      stage('Push Docker Image to Docker Registry') {          container('docker'){              withCredentials([[$class: 'UsernamePasswordMultiBinding',              credentialsId: env.DOCKER\_CREDENTIALS\_ID,              usernameVariable: 'USERNAME',              passwordVariable: 'PASSWORD']]) {                  docker.withRegistry(env.DOCEKR\_REGISTRY, env.DOCKER\_CREDENTIALS\_ID) {                      sh("docker push ${app1\_image\_tag}")                      sh("docker push ${app2\_image\_tag}")                  }              }          }      } |

* After previous stages have competed successfully, a deployment to kubernetes is triggered using the “kubectl” container. Note that the values of the K8s environment variables should have been configured in a previous step.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | //Stage 7: Deploy Application on K8s      stage('Deploy Application on K8s') {          container('kubectl'){              withKubeConfig([credentialsId: env.K8s\_CREDENTIALS\_ID,              serverUrl: env.K8s\_SERVER\_URL,              contextName: env.K8s\_CONTEXT\_NAME,              clusterName: env.K8s\_CLUSTER\_NAME]){                  sh("kubectl apply -f configmap.yml")                  sh("kubectl apply -f secret.yml")                  sh("kubectl apply -f postgres.yml")                  sh("kubectl apply -f ${app1\_name}.yml")                  sh("kubectl set image deployment/${app1\_name} ${app1\_container\_name}=${app1\_image\_tag}")                  sh("kubectl apply -f ${app2\_name}.yml")                  sh("kubectl set image deployment/${app2\_name} ${app2\_container\_name}=${app2\_image\_tag}")              }          }     } |

With that, you are ready to go! You can either change something in your Github repository and commit it, then the build of the pipeline will start automatically in jenkins, or you can start the build manually in jenkins UI. After it finishes, you should see a view similar to this, in which you can see the details of each stage (time it took, success or failure, logs) along with the Console Output of the build.

**Summary**

There you go! In this Blogpost, you have successfully created a CI/CD pipeline for a containerized spring-boot java application on Kubernetes using *Jenkins*. In Part 2 of this Blog-Series which will follow soon, we will set up a CI/CD pipeline using Bitbucket as a tool.

Top of Form



Bottom of Form

* [Cloud Computing](https://www.novatec-gmbh.de/en/blog/category/architecture-and-cloud/cloud-computing-en/)
* [Continuous Integration](https://www.novatec-gmbh.de/en/blog/category/development-programming-en/development-en/continuous-integration-en/)
* [Development & Programming](https://www.novatec-gmbh.de/en/blog/category/development-programming-en/)
* [DevOps](https://www.novatec-gmbh.de/en/blog/category/quality-engineering-en/devops-en/)
* [Education](https://www.novatec-gmbh.de/en/blog/category/education-en/)

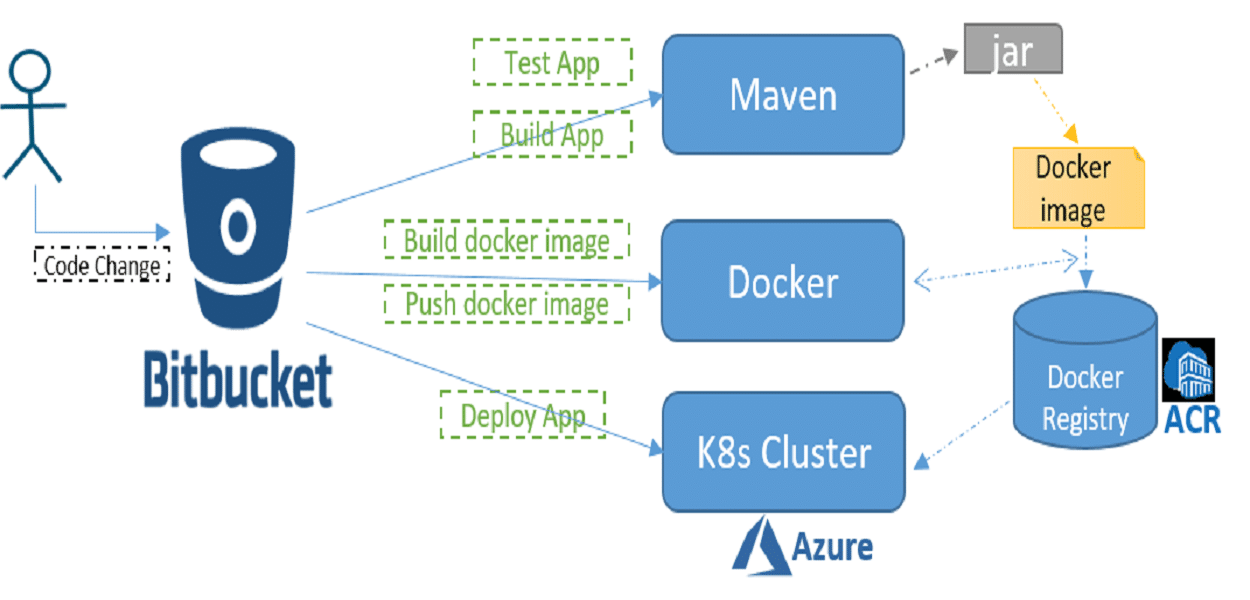
### Related posts

[Debugging on Kubernetes - the perfect developer machine](https://www.novatec-gmbh.de/en/blog/debugging-on-kubernetes-the-perfect-developer-machine/)

[Read more](https://www.novatec-gmbh.de/en/blog/debugging-on-kubernetes-the-perfect-developer-machine/)

#### [Create a REST Application with kTor and Exposed](https://www.novatec-gmbh.de/en/blog/creating-a-rest-application-with-ktor-and-exposed/)

[Read more](https://www.novatec-gmbh.de/en/blog/creating-a-rest-application-with-ktor-and-exposed/)

[[](https://www.novatec-gmbh.de/en/blog/ci-cd-series-part-2-how-to-build-a-ci-cd-pipeline-using-bitbucket-docker-kubernetes-step-by-step-guide/)](https://www.novatec-gmbh.de/en/blog/ci-cd-series-part-2-how-to-build-a-ci-cd-pipeline-using-bitbucket-docker-kubernetes-step-by-step-guide/)

#### [CI/CD Series - Part 2: How to build a CI/CD pipeline using Bitbucket, ...](https://www.novatec-gmbh.de/en/blog/ci-cd-series-part-2-how-to-build-a-ci-cd-pipeline-using-bitbucket-docker-kubernetes-step-by-step-guide/)

[Read more](https://www.novatec-gmbh.de/en/blog/ci-cd-series-part-2-how-to-build-a-ci-cd-pipeline-using-bitbucket-docker-kubernetes-step-by-step-guide/)

#### [Deployment of a Spring Boot application using AWS Elastic Beanstalk, A...](https://www.novatec-gmbh.de/en/blog/deployment-of-a-spring-boot-application-using-aws-elastic-beanstalk-amazon-route-53-and-aws-certificate-manager/)

[Read more](https://www.novatec-gmbh.de/en/blog/deployment-of-a-spring-boot-application-using-aws-elastic-beanstalk-amazon-route-53-and-aws-certificate-manager/)

ANSIBLE TEMPATE FOR INFASTRUCTURE ANS CODE DEPLOYMENT FROM MASTER SERVER TO SLAVE SERVERS FOR DEPLOYMENT AND AUTOMATION

# template – Template a file out to a remote server[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#template-template-a-file-out-to-a-remote-server)

* [Synopsis](https://docs.ansible.com/ansible/latest/modules/template_module.html#synopsis)
* [Parameters](https://docs.ansible.com/ansible/latest/modules/template_module.html#parameters)
* [Notes](https://docs.ansible.com/ansible/latest/modules/template_module.html#notes)
* [See Also](https://docs.ansible.com/ansible/latest/modules/template_module.html#see-also)
* [Examples](https://docs.ansible.com/ansible/latest/modules/template_module.html#examples)
* [Status](https://docs.ansible.com/ansible/latest/modules/template_module.html#status)

## [Synopsis](https://docs.ansible.com/ansible/latest/modules/template_module.html#id1)[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#synopsis)

* Templates are processed by the [Jinja2 templating language](http://jinja.pocoo.org/docs/).
* Documentation on the template formatting can be found in the [Template Designer Documentation](http://jinja.pocoo.org/docs/templates/).
* Additional variables listed below can be used in templates.
* ansible\_managed (configurable via the defaults section of ansible.cfg) contains a string which can be used to describe the template name, host, modification time of the template file and the owner uid.
* template\_host contains the node name of the template’s machine.
* template\_uid is the numeric user id of the owner.
* template\_path is the path of the template.
* template\_fullpath is the absolute path of the template.
* template\_destpath is the path of the template on the remote system (added in 2.8).
* template\_run\_date is the date that the template was rendered.

## [Parameters](https://docs.ansible.com/ansible/latest/modules/template_module.html#id2)[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#parameters)

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Choices/Defaults** | **Comments** |
| **attributes**  string |  | The attributes the resulting file or directory should have.  To get supported flags look at the man page for chattr on the target system.  This string should contain the attributes in the same order as the one displayed by lsattr.  The = operator is assumed as default, otherwise + or - operators need to be included in the string.  aliases: attr |
| **backup**  boolean | **Choices:**   * **no** ← * yes | Create a backup file including the timestamp information so you can get the original file back if you somehow clobbered it incorrectly. |
| **block\_end\_string**  string  *added in 2.4* | **Default:**  "%}" | The string marking the end of a block. |
| **block\_start\_string**  string  *added in 2.4* | **Default:**  "{%" | The string marking the beginning of a block. |
| **dest**  path / required |  | Location to render the template to on the remote machine. |
| **follow**  boolean  *added in 2.4* | **Choices:**   * **no** ← * yes | Determine whether symbolic links should be followed.  When set to yes symbolic links will be followed, if they exist.  When set to no symbolic links will not be followed.  Previous to Ansible 2.4, this was hardcoded as yes. |
| **force**  boolean | **Choices:**   * no * **yes** ← | Determine when the file is being transferred if the destination already exists.  When set to yes, replace the remote file when contents are different than the source.  When set to no, the file will only be transferred if the destination does not exist. |
| **group**  string |  | Name of the group that should own the file/directory, as would be fed to chown. |
| **lstrip\_blocks**  boolean  *added in 2.6* | **Choices:**   * **no** ← * yes | Determine when leading spaces and tabs should be stripped.  When set to yes leading spaces and tabs are stripped from the start of a line to a block.  This functionality requires Jinja 2.7 or newer. |
| **mode**  string |  | The permissions the resulting file or directory should have.  For those used to /usr/bin/chmod remember that modes are actually octal numbers. You must either add a leading zero so that Ansible's YAML parser knows it is an octal number (like 0644 or 01777) or quote it (like '644' or '1777') so Ansible receives a string and can do its own conversion from string into number.  Giving Ansible a number without following one of these rules will end up with a decimal number which will have unexpected results.  As of Ansible 1.8, the mode may be specified as a symbolic mode (for example, u+rwx or u=rw,g=r,o=r).  As of Ansible 2.6, the mode may also be the special string preserve.  When set to preserve the file will be given the same permissions as the source file. |
| **newline\_sequence**  string  *added in 2.4* | **Choices:**   * **\n** ← * \r * \r\n | Specify the newline sequence to use for templating files. |
| **output\_encoding**  string  *added in 2.7* | **Default:**  "utf-8" | Overrides the encoding used to write the template file defined by dest.  It defaults to utf-8, but any encoding supported by python can be used.  The source template file must always be encoded using utf-8, for homogeneity. |
| **owner**  string |  | Name of the user that should own the file/directory, as would be fed to chown. |
| **selevel**  string | **Default:**  "s0" | The level part of the SELinux file context.  This is the MLS/MCS attribute, sometimes known as the range.  When set to \_default, it will use the level portion of the policy if available. |
| **serole**  string |  | The role part of the SELinux file context.  When set to \_default, it will use the role portion of the policy if available. |
| **setype**  string |  | The type part of the SELinux file context.  When set to \_default, it will use the type portion of the policy if available. |
| **seuser**  string |  | The user part of the SELinux file context.  By default it uses the system policy, where applicable.  When set to \_default, it will use the user portion of the policy if available. |
| **src**  path / required |  | Path of a Jinja2 formatted template on the Ansible controller.  This can be a relative or an absolute path.  The file must be encoded with utf-8 but output\_encoding can be used to control the encoding of the output template. |
| **trim\_blocks**  boolean  *added in 2.4* | **Choices:**   * no * **yes** ← | Determine when newlines should be removed from blocks.  When set to yes the first newline after a block is removed (block, not variable tag!). |
| **unsafe\_writes**  boolean | **Choices:**   * **no** ← * yes | Influence when to use atomic operation to prevent data corruption or inconsistent reads from the target file.  By default this module uses atomic operations to prevent data corruption or inconsistent reads from the target files, but sometimes systems are configured or just broken in ways that prevent this. One example is docker mounted files, which cannot be updated atomically from inside the container and can only be written in an unsafe manner.  This option allows Ansible to fall back to unsafe methods of updating files when atomic operations fail (however, it doesn't force Ansible to perform unsafe writes).  IMPORTANT! Unsafe writes are subject to race conditions and can lead to data corruption. |
| **validate**  string |  | The validation command to run before copying into place.  The path to the file to validate is passed in via '%s' which must be present as in the examples below.  The command is passed securely so shell features like expansion and pipes will not work. |
| **variable\_end\_string**  string  *added in 2.4* | **Default:**  "}}" | The string marking the end of a print statement. |
| **variable\_start\_string**  string  *added in 2.4* | **Default:**  "{{" | The string marking the beginning of a print statement. |

## [Notes](https://docs.ansible.com/ansible/latest/modules/template_module.html#id3)[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#notes)

Note

* You can use the [copy](https://docs.ansible.com/ansible/latest/modules/copy_module.html#copy-module) module with the content: option if you prefer the template inline, as part of the playbook.
* For Windows you can use [win\_template](https://docs.ansible.com/ansible/latest/modules/win_template_module.html#win-template-module) which uses ‘\r\n’ as newline\_sequence by default.
* Including a string that uses a date in the template will result in the template being marked ‘changed’ each time.
* Since Ansible 0.9, templates are loaded with trim\_blocks=True.
* Also, you can override jinja2 settings by adding a special header to template file. i.e. #jinja2:variable\_start\_string:'[%', variable\_end\_string:'%]', trim\_blocks: False which changes the variable interpolation markers to [% var %] instead of {{ var }}. This is the best way to prevent evaluation of things that look like, but should not be Jinja2.
* Using raw/endraw in Jinja2 will not work as you expect because templates in Ansible are recursively evaluated.
* To find Byte Order Marks in files, use Format-Hex <file> -Count 16 on Windows, and use od -a -t x1 -N 16 <file> on Linux.

## [See Also](https://docs.ansible.com/ansible/latest/modules/template_module.html#id4)[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#see-also)

See also

[**copy – Copy files to remote locations**](https://docs.ansible.com/ansible/latest/modules/copy_module.html#copy-module)

The official documentation on the **copy** module.

[**win\_copy – Copies files to remote locations on windows hosts**](https://docs.ansible.com/ansible/latest/modules/win_copy_module.html#win-copy-module)

The official documentation on the **win\_copy** module.

[**win\_template – Template a file out to a remote server**](https://docs.ansible.com/ansible/latest/modules/win_template_module.html#win-template-module)

The official documentation on the **win\_template** module.

## [Examples](https://docs.ansible.com/ansible/latest/modules/template_module.html#id5)[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#examples)

- name: Template a file to /etc/files.conf

template:

src: /mytemplates/foo.j2

dest: /etc/file.conf

owner: bin

group: wheel

mode: '0644'

- name: Template a file, using symbolic modes (equivalent to 0644)

template:

src: /mytemplates/foo.j2

dest: /etc/file.conf

owner: bin

group: wheel

mode: u=rw,g=r,o=r

- name: Copy a version of named.conf that is dependent on the OS. setype obtained by doing ls -Z /etc/named.conf on original file

template:

src: named.conf\_{{ ansible\_os\_family }}.j2

dest: /etc/named.conf

group: named

setype: named\_conf\_t

mode: 0640

- name: Create a DOS-style text file from a template

template:

src: config.ini.j2

dest: /share/windows/config.ini

newline\_sequence: '\r\n'

- name: Copy a new sudoers file into place, after passing validation with visudo

template:

src: /mine/sudoers

dest: /etc/sudoers

validate: /usr/sbin/visudo -cf %s

- name: Update sshd configuration safely, avoid locking yourself out

template:

src: etc/ssh/sshd\_config.j2

dest: /etc/ssh/sshd\_config

owner: root

group: root

mode: '0600'

validate: /usr/sbin/sshd -t -f %s

backup: yes

## [Status](https://docs.ansible.com/ansible/latest/modules/template_module.html#id6)[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#status)

* This module is guaranteed to have backward compatible interface changes going forward. [stableinterface]
* This module is [maintained by the Ansible Core Team](https://docs.ansible.com/ansible/latest/user_guide/modules_support.html#modules-support). [core]

### Red Hat Support[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#red-hat-support)

More information about Red Hat’s support of this module is available from this [Red Hat Knowledge Base article](https://access.redhat.com/articles/3166901).

### Authors[¶](https://docs.ansible.com/ansible/latest/modules/template_module.html#authors)

* Ansible Core Team
* Michael DeHaan

Hint

If you notice any issues in this documentation, you can [edit this document](https://github.com/ansible/ansible/edit/devel/lib/ansible/modules/files/template.py?description=%23%23%23%23%23%20SUMMARY%0A%3C!---%20Your%20description%20here%20--%3E%0A%0A%0A%23%23%23%23%23%20ISSUE%20TYPE%0A-%20Docs%20Pull%20Request%0A%0A%2Blabel:%20docsite_pr) to improve it.

© Copyright 2019 Red Hat, Inc. Last updated on Apr 17, 2020.

[Serverwise: the SSD Nodes blog](https://blog.ssdnodes.com/blog)

[Tutorials](https://blog.ssdnodes.com/blog/category/tutorials/) [Self-hosting](https://blog.ssdnodes.com/blog/category/self-hosting/) [Comparisons](https://blog.ssdnodes.com/blog/category/comparisons/) [Updates](https://blog.ssdnodes.com/blog/category/updates/) [VPS Uses](https://blog.ssdnodes.com/blog/category/uses/) [Development](https://blog.ssdnodes.com/blog/category/development/) [SaaS Alternatives](https://blog.ssdnodes.com/blog/category/alternatives/)

[Tutorials](https://blog.ssdnodes.com/blog/category/tutorials/) | Aug 23, 2018 | 38 min read

**ANSIBEL FOR INFASTRUCTURE AS CODE AND SERVER AUTOMATION**

# A step by step guide to Ansible (Tutorial)

**<img src="https://blog.ssdnodes.com/blog/wp-content/uploads/2018/05/dd-120x120.jpg" width="96" height="96" alt="Dwijadas Dey" class="avatar avatar-96 wp-user-avatar wp-user-avatar-96 alignnone photo" /> Written by**[**Dwijadas Dey**](https://blog.ssdnodes.com/blog/author/dd/)

<img width="1440" height="600" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2018/08/201808\_step-by-step-ansible-guide-1.jpeg" class=" wp-post-image" alt="Step-by-step Ansible guide, from installation to playbooks" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2018/08/201808\_step-by-step-ansible-guide-1.jpeg 2880w, https://blog.ssdnodes.com/blog/wp-content/uploads/2018/08/201808\_step-by-step-ansible-guide-1-250x104.jpeg 250w, https://blog.ssdnodes.com/blog/wp-content/uploads/2018/08/201808\_step-by-step-ansible-guide-1-768x320.jpeg 768w, https://blog.ssdnodes.com/blog/wp-content/uploads/2018/08/201808\_step-by-step-ansible-guide-1-700x292.jpeg 700w, https://blog.ssdnodes.com/blog/wp-content/uploads/2018/08/201808\_step-by-step-ansible-guide-1-120x50.jpeg 120w" sizes="(max-width: 1440px) 100vw, 1440px" />

In today’s Ansible for beginners tutorial, we’re going to guide you through everything you need to know in one sitting– from installing Ansible on your machine to setting up and running an example Playbook.

Then we’ll talk a bit about how to best put it to use on your VPS. Or multiple VPS’s.

Your first steps with Ansible might be shaky, but once you master its many options and complexities, you’ll open yourself up to an entirely new level of power when configuring servers.

[Now in Europe! 🌍](https://ssdnodes.com/sale/europe/?blogbanner" \t "_blank)

[Get THE BEST DEALS IN CLOUD HOSTING from Frankfurt!](https://ssdnodes.com/sale/europe/?blogbanner" \t "_blank)

**[Grab a huge 32GB RAM & 320GB of SSD storage for just $109/year!](https://ssdnodes.com/sale/europe/?blogbanner" \t "_blank)**

So let’s get started with our comprehensive, step-by-step Ansible guide!

## Notes on the Ansible tutorial:

* This tutorial requires the use of domain names. Whenever you see either the SUBDOMAIN, DOMAIN, or TLD variables, replace them with the details of your domain name. In example.ssdnodes.com, example is the SUBDOMAIN, ssdnodes is the DOMAIN, and .com is the TLD.
* This tutorial requires the use of IP address. Whenever you see the IP\_ADDRESS variable, replace it with your own.

## What is “Infrastructure As Code”?

**Infrastructure as code** (IaC) is the way of defining computing and network infrastructure through source code, the same way you do for applications. Rather than manually configuring your infrastructure or using a one-off isolated script, IaC gives you the power to write code, using a high-level language, to decide how infrastructure should be configured and deployed.

IaC is different from infrastructure automation, which involves repeating the steps multiple times and spawning them on several servers.

The guiding principle behind IaC is to enforce consistency among DevOps team members by representing the desired state of their infrastructure via code. Moreover, the code can be kept in source control, which means it can be audited, tested on, and used to create reproducible builds with continuous delivery.

## What is Ansible?

[Ansible](https://www.ansible.com/) is an open source IT configuration management, deployment, and orchestration tool. It empowers DevOps teams to define their infrastructure as a code in a simple and declarative manner.

A lot of people compare Ansible to similar tools like [Chef](https://www.chef.io/chef/) or [Puppet](https://puppet.com/). They all help automate and provision infrastructure, but there are a few features that make me prefer Ansible over the others.

## Why use Ansible?

### Ansible is agentless

Ansible doesn’t need any agents to be installed on remote systems to be managed, which means less maintenance overhead and performance issues. Instead, Ansible uses a push-based approach leveraging existing SSH connections to run tasks on the remote managed host. Chef or Puppet work by installing an agent on the hosts to be managed and the agent pulls changes from the control host using its own channel.

### It’s written in Python

Ansible is written in Python, which means installing and running Ansible in any Linux distribution is very easy, and only a little more difficult on OS X. Being a popular language, there’s also a good chance that you’re familiar with it, or at least can find enough resources online to start learning. Or, you’ll easily be able to find a developer with Python experience to help you out.

### Learn Ansible in minutes

The fact that a new user can get up to speed and run Ansible tasks in a matter of minutes, thanks to clear and easy-to-follow documentation, is one of the most appealing features of Ansible. Troubleshooting in Ansible is also very easy for beginners, and the fact that all tasks are idempotent reduces the risk of making a mistake.

### Deploy infrastructure in record time

Ansible can dispatch tasks to multiple remote managed hosts in parallel. This means you can execute Ansible tasks on a second managed host without waiting for them to complete on the first to reduce provision time and deploy your infrastructure faster than ever.

**What’s the BEST DEAL in cloud hosting?**

Develop at hyperspeed with a Performance VPS from SSD Nodes. We DOUBLED the amount of blazing-fast NVMe storage on our most popular plan and beefed up the CPU offering on these plans. There’s nothing else like it on the market, at least not at these prices.

**Score a 16GB Performance VPS with 160GB of NVMe storage for just $89/year for a limited time!**

[**Get limited-time deals!⚡**](https://www.ssdnodes.com/)

## Step 1: Install Ansible on your control machine

To take your first steps with Ansible, you first need to install it on your **control machine**. This is the machine you’ll use to dispatch tasks. For most people, this will be your desktop machine at home or your laptop, but you can also use one VPS as a control host to connect to other VPSs.

### Installing Ansible on Ubuntu 16.04

You can install Ansible using standard package managers like apt/yum or Python’s pip command. To install it using standard package manager in Ubuntu, add its repository information apt-add-repository. Next, update the system and install Ansible using apt-get.

$ sudo apt-get install software-properties-common

$ sudo apt-add-repository ppa:ansible/ansible

$ sudo apt-get update

$ sudo apt-get install ansible

### Installing Ansible on CentOS 7

While installing Ansible in CentOS and RHEL, you need to enable the EPEL repository first before proceeding with installation of Ansible. Once you enabled the EPEL repository, install Ansible using yum.

$ cd /tmp

$ wget http://dl.fedoraproject.org/pub/epel/7/x86\_64/e/epel-release-7-9.noarch.rpm

$ sudo rpm -ivh epel-release-7-9.noarch.rpm

$ sudo yum update

$ sudo yum install ansible

## Guide to working with Ansible

You are now up and running with Ansible in your system, which is otherwise known as the **control host**. The control host is the Ansible host you to dispatch tasks to the remote managed Ansible hosts.

Before you start delegating tasks to a managed host, make sure you have non-root, a sudo-enabled user on that host—it’s always a bad idea to connect to a remote VPS via a root user.

### Ansible inventory files

The Ansible inventory file lists which hosts will receive commands from the control host. The inventory can list individual hosts, or group them under categories you distinguish.

The default location for the inventory file is /etc/ansible/hosts, but it’s also possible to change the location of the inventory file by uncommenting the inventory parameter in /etc/ansible/ansible.cfg

A typical inventory file can list the managed host either by IP address or by domain names. It is also possible to list one managed host in more than one group. Here’s an example of listing two hosts under the webservers and dbservers categories.

[webservers]

123.45.67.89

SUBDOMAIN.DOMAIN.TLD

[dbservers]

123.45.67.89

SUBDOMAIN.DOMAIN.TLD

To test if all the hosts are discoverable by the inventory file, use the following ad-hoc command.

$ ansible all --list-hosts

hosts (2):

123.45.67.89

SUBDOMAIN.DOMAIN.TLD

You can also list the hosts by group name:

$ ansible dbservers --list-hosts

hosts (2):

123.45.67.89

SUBDOMAIN.DOMAIN.TLD

**Ad-hoc** commands in Ansible are merely those that perform a single command across one or many hosts. They don’t use tasks but allow you to do a lot of things quite easily without building out playbooks (more on those in the second part of this guide).

To find out if all the hosts are up and running, use the following ad-hoc command that uses the ping module of Ansible. The -u switch specifies which user Ansible will connect to via SSH—change it according to the non-root user you created earlier.

$ ansible all -m ping -u USER

123.45.67.89 | SUCCESS => {

"changed": false,

"ping": "pong"

}

...

...

The "changed": false in the above JSON result tells us that the ping Ansible task didn’t change anything on the remote server.

Rather than specifying all the hosts as in the above command, you can also ping a group of hosts. Specify the group name in place of ‘all’ with the following command:

$ ansible webservers -m ping -u USER

### Ansible modules

**Modules** are the discrete units of code that can be used from the terminal or in a playbook task. They simplify Ansible tasks by installing software, copying files, using templates, and so on.

Modules use the available context to determine what actions if any needed to bring the managed host to the desired state and are idempotent, that means if you run the same task again and again, the state of the machine will not change.

To find the list of available modules, use the following command:

$ ansible-doc -l

Let’s try to install Nginx on an Ubuntu/Debian host using an ad-hoc command in Ansible:

$ ansible webservers -b --become-user=root -m shell -a 'apt -y install nginx' -u USER

172.104.160.8 | SUCCESS | rc=0 >>

Reading package lists...

Building dependency tree...

The following flags were used with the above command:

* -b: Instruct ansible to become another user to run the command
* --become-user=root: Run the command as a root user
* -m: Declares which module is used in the command
* -a: Declares which arguments are passed to the module

The alternate and preferred way of installing software using an ad-hoc command is to use apt module. If your remote managed host is running RHEL/CentOS, then change the module name from apt to yum.

$ ansible webservers -b --become-user=root -m apt -a 'name=nginx state=present update\_cache=true' -u ansadm

172.104.160.8 | SUCCESS => {

"cache\_update\_time": 1530378409,

"cache\_updated": true,

"changed": true,

"stderr": "",

"stderr\_lines": [],

...

...

In the above Ansible command, the -a switch passes the arguments to the apt module by specifying the name of the package to be installed, the desired state, and whether to update the package repository cache or not.

The line change: true in the result section of the above ad-hoc command signifies that the state of the system has been changed. If you run the above ad-hoc command again, the value of changed field will be false, which means the state of the system remains unchanged, because Ansible is aware that Nginx is already present in the system and will not try to alter the state again.

That’s what we call Ansible **idempotent**. You can run the same ad-hoc command as many times as you’d like and it won’t change anything unless it needs to.

172.104.160.8 | SUCCESS => {

"cache\_update\_time": 1530378676,

"cache\_updated": true,

"changed": false

}

So far, we have understood the ansible modules and its usages through ad-hoc way, but this is not so useful until we use the modules in ansible playbooks to run multiple tasks in the remote managed host.

### Tasks in Ansible

When you dispatch a job from a control host to a managed host using one an Ansible module, it is known as a **task**. Tasks can be implemented using ad-hoc commands, as we’ve done just above, or you can use an Ansible **playbook** (more on those in a moment).

One example of a task is copying a file from the control host to a managed host, since it requires the use of ‘copy’ module. There are thousands of modules in Ansible, which means a task can use any of the modules to bring a managed host to the desired state. How many modules are there by default in Ansible? Let’s see:

$ ansible-doc -l | wc -l

1852

If you haven’t guessed, there are a lot of things you can do when combining Ansible tasks and modules.

### Play in Ansible

An Ansible **play** is a set of tasks that are run on one or more managed hosts. A play may include one or many different tasks, and the most common way to execute a play is to use a playbook.

### Ansible Playbooks

No Ansible tutorial would be complete without a guide to Playbooks. And some concrete Ansible Playbook examples.

Ansible **Playbooks** are composed of one or more plays and offer more advanced functionality for sending tasks to managed host compared to running many ad-hoc commands.

The tasks in Ansible playbooks are written in Yet Another Markup Language (YAML), which is easier to understand than a JSON or XML file. Each task in the playbook is executed sequentially for each host in the inventory file before moving on to the next task.

Let’s create a simple Ansible playbook example that will install Nginx and a MySQL server on the managed hosts that we had already defined in the inventory file.

To be more precise, we want Nginx installed on hosts in the webservers group and a MySQL server installed on hosts in the dbservers group.

$ vi playbook.yml

---

- hosts: webservers

gather\_facts: yes

become\_user: root

tasks:

- name: Install Nginx

apt: pkg=nginx state=present

notify:

- restart nginx

- name: Enable Nginx during boot

service: name=nginx state=started enabled=yes

handlers:

- name: restart nginx

service: name=nginx state=restarted

- hosts: dbservers

become\_user: root

tasks:

- name: Install mysql

apt: pkg=mysql-server state=present

The hosts tells Ansible on which hosts to run the tasks. The above Ansible playbook includes two host groups from the inventory file. The tasks for webservers group are to install Nginx and enable Nginx during boot, and the dbservers group includes a single task to install MySQL.

The become\_user in both the host section tells ansible to use sudo to run the tasks.

The gather\_facts option gathers information about managed hosts such as distribution, OS family, and more. In ansible terminology, this information is known as FACTS.

The handlers section restarts Nginx when Ansible gets notified that Nginx has been installed.

A handler is the same as a task, but it will be executed when called by another task. It is like an event-driven system. A handler will run a task only when it is called by an event it listens for.

Now run the above playbook example using ansible-playbook. Append the name of the user from a remote managed host in the command using -u switch.

$ ansible-playbook playbook.yml -u USER

PLAY [webservers] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [123.45.67.89]

TASK [Install Nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [123.45.67.89]

TASK [Enable Nginx during boot] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [123.45.67.89]

RUNNING HANDLER [restart nginx] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [123.45.67.89]

PLAY [dbservers] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [123.45.67.89]

TASK [Install mysql] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [123.45.67.89]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

123.45.67.89 : ok=6 changed=3 unreachable=0 failed=0

The last line contains information about the current run of the above playbook. The four points of data are:

* ok: The number of tasks that were either executed correctly or didn’t result in a change.
* changed: The number of things that were modified by Ansible.
* unreachable: The number of hosts that were unreachable for some reason.
* failed: The number of tasks failed to execute correctly.

### Roles

In Ansible, a **role** provides a mechanism to break a complicated playbook into multiple reusable components. Each component offers a small function that can be used independently within the playbook. So rather than creating one complex playbook, you can create many roles and simply drop them into your playbooks.

You can’t execute roles directly, the way you do a playbook, and you can’t specify which host you want to execute a role, the way you would an ad-hoc command. Instead, they’re built into the playbooks you use to define a host.

The [Ansible Galaxy](https://galaxy.ansible.com/) repository has thousands of pre-built roles for you to choose from, although you’re free to create your role framework. Let’s dig into how you might want to do just that.

### Variables

In Ansible, **variables** are similar to variables in any programming language—they let you input values and numbers dynamically into your playbook. Variables simplify operations by allowing you define and declare them throughout all the various roles and tasks you want to perform.

There are few places where you can define variables in an Ansible playbook.

* In the playbook
* In the inventory file
* In a separate variable file
* Using group\_vars

To define variables in a playbook, use vars key just above the task where you want to use the variable. Once declared, you can use it inside the {{ }} tag. Let’s declare a variable by the name pkgname and assign it the value of the package name that we want to install, which is nginx. Once done, we can use the variable in a task.

---

- hosts: webservers

gather\_facts: yes

become\_user: root

vars:

pkgname: nginx

tasks:

- name: Install "{{ pkgname }}"

apt: pkg="{{ pkgname }}" state=present

...

...

It is also possible to declare a variable in the inventory file using the syntax [host\_group\_name:vars]. Let’s define the variable pkgname in the inventory file.

[webservers:vars]

pkgname=nginx

Now the variable pkgname can be used anywhere in the webservers hosts section in the playbook.

You can also define variables in a separate variable file and import it into the playbook. Create a variable file using vi another text editor and define the variable pkgname here.

$ vi ansible\_vars.yml

---

pkgname: nginx

To use the variable pkgname, import the above file using the vars\_files keyword in the playbook.

$ vi playbook.yml

---

- hosts: webservers

gather\_facts: yes

become\_user: root

vars\_files:

- ./ansible\_vars.yml

...

...

Another preferred way of managing variables is to create a group\_vars directory inside your Ansible working directory. Ansible will load any YAML files in this directory with the name of any Ansible group.

Create the directory group\_vars in your Ansible working directory, and then create the variable files matching with the group name from the inventory file. In our example, this would be webservers and dbservers. This allows you to separate variables according to host groups, which can make everything easier to manage.

$ cd <your\_ansible\_working\_directory>

$ mkdir group\_vars

$ cd group\_vars

$ vi webservers

---

pkgname: nginx

$ vi dbservers

---

pkgname: mysql-server

You don’t need to declare the variable in your playbook, as Ansible will automatically pull the variables from each group\_vars files and will substitute them during runtime.

Now suppose you want to have variables that will apply to all the host groups mentioned in the inventory file. To accomplish it, name a file by the name all inside group\_vars directory. The group\_vars/all files are used to set variables for every host that Ansible connects to.

### Conditionals

In Ansible, **conditionals** are analogous to an if statement in any programming language. You use a conditional when you want to execute a task based on certain conditions.

In our last playbook example, we installed Nginx, so let’s extend that by creating a task that installs Nginx when Apache is not present on the host. We can add another task to the playbook we’ve already built.

...

...

tasks:

- name: Check if Apache is already installed

shell: dpkg -s apache2 | grep Status

register: apache2\_is\_installed

failed\_when: no

- name: Install "{{ pkgname }}"

apt: pkg="{{ pkgname }}" state=present

when: apache2\_is\_installed.rc == 1

notify:

- restart nginx

...

...

The first task in the above playbook checks if Apache is installed using dpkg -s command and stores the output of the task to apache2\_is\_installed variable. The return value of the task will be a non-zero value if Apache is not installed on the host.

Usually, Ansible would stop executing other tasks because of this non-zero value, but the failed\_when: no gives Ansible permission to continue with the next set of tasks when it encounters a non-zero value.

The second task will install Nginx only when the return value of rc is equal to one, which is declared via when: apache2\_is\_installed.rc == 1.

### Loops

All programming languages provide a way to iterate over data to perform some repetitive task. Ansible also provides a way to do the same using a concept called **looping**, which is supplied by Ansible lookup plugins. With loops, a single task in one playbook can be used to create multiple users, install many packages, and more.

While there are many ways to use loops in Ansible, we’ll cover just one of them to get you started. The easiest way to use loops in ansible is to use with\_items keyword, which is used to iterate over an item list to perform some repetitive tasks. The following playbook includes a task which installs packages in a loop using the keyword with\_items.

---

- hosts: webservers

gather\_facts: yes

become\_user: root

tasks:

- name: Installing packages using loops

apt: pkg={{ item }} state=present update\_cache=yes

with\_items:

- sysstat

- htop

- git

Run the above playbook from your command line, and you’ll see that you’ve installed all three packages on the remote host with a single task!

### Tags

**Tags** allow you to run only specific tasks from your playbook via the command line. Just add the tags keyword for each task and run only the task(s) that you want by using --tags switch at the end of the ansible command. In the following playbook, we have added tags at the end of each task, thereby allowing us to run tasks separately from a single playbook.

---

- hosts: webservers

gather\_facts: yes

become\_user: root

tasks:

- name: Check if Apache is already installed

shell: dpkg -s apache2 | grep Status

register: apache2\_is\_installed

failed\_when: no

- name: Install "{{ pkgname }}"

apt: pkg="{{ pkgname }}" state=present

when: apache2\_is\_installed.rc == 1

notify:

- restart nginx

- name: ensure nginx is running and enable it at boot

service: name=nginx state=started enabled=yes

tags:

- mytag1

handlers:

- name: restart nginx

service: name=nginx state=restarted

- hosts: dbservers

become\_user: root

tasks:

- name: Install mysql

apt: pkg="{{ pkgname }}" state=present

tags:

- mytag2

Now run any of the tasks by specifying tag name at the end of ansible command.

$ ansible-playbook playbook.yml -u ansadm --tags 'mytag2'

### How to use Ansible templates

Typically, after installing a web server like Nginx, you need to configure a virtual hosts file to properly serve a given website on your VPS. Instead of using SSH to log into your VPS to configure it after running Ansible, or using Ansible’s copy module to copy many unique configuration files individually, you can take advantage of Ansible’s **templates** features.

A template file contains all of the configuration parameters you need, such as the Nginx virtual host settings, and uses variables, which are replaced by the appropriate values when the playbook is executed. Template files usually end with the .j2 extension that denotes the Jinja2 templating engine.

To begin working with templates, create a directory for template files in your Ansible working directory.

$ mkdir templates

Create two template files. The first template file will be the default index.html file for each site, and the second template file will contain configuration settings for the Nginx virtual host.

$ cd templates

$ vi index.html.j2

<html>

You are visiting {{ domain\_name }} !

</html>

Similarly, create a template file for the Nginx virtual host:

$ vi nginx-vh.j2

server {

listen 80;

server\_name {{ domain\_name }};

client\_max\_body\_size 20m;

index index.php index.html index.htm;

root /var/www/html/{{ domain\_name }};

location / {

try\_files $uri $uri/ /index.html?q=$uri&$args;

}

location ~\* \.(js|css|png|jpg|jpeg|gif|ico|woff|ttf|svg|otf)$ {

expires 30d;

add\_header Pragma public;

add\_header Cache-Control "public";

access\_log off;

}

}

Notice that the variables domain\_name in the above two template files are enclosed within {{ }}, which means they will be substituted during runtime by the value of this variable. To define the variable domain\_name, navigate to the group\_vars directory and edit the file webservers and add the following lines in it.

$ cd group\_vars

$ vi webservers

---

domain\_name: SUBDOMAIN.DOMAIN.TLD

Finally, edit the ansible playbook to create a root folder for sites, copy the index.html file to the site’s root folder, and copy the virtual host file to the Nginx virtual host directory /etc/nginx/sites-enabled one by one.

$ vi playbook.yml

---

- hosts: webservers

gather\_facts: yes

become\_user: root

tasks:

- name: Check if Apache is already installed

shell: dpkg -s apache2 | grep Status

register: apache2\_is\_installed

failed\_when: no

- name: Install "{{ pkgname }}"

apt: pkg="{{ pkgname }}" state=present

when: apache2\_is\_installed.rc == 1

notify:

- restart nginx

- name: ensure nginx is running and enable it at boot

service: name=nginx state=started enabled=yes

- name: create virtual host root directory

file: name=/var/www/html/{{ domain\_name }} state=directory

- name: Copying index file to webroot

template:

src: templates/index.html.j2

dest: /var/www/html/{{ domain\_name }}/index.html

- name: Enables nginx virtual host

template:

src: templates/nginx-vh.j2

dest: /etc/nginx/sites-enabled/{{ domain\_name }}

- name: restart nginx

service: name=nginx state=restarted

tags:

- mytag1

handlers:

- name: restart nginx

service: name=nginx state=restarted

- hosts: dbservers

become\_user: root

tasks:

- name: Install mysql

apt: pkg="{{ pkgname }}" state=present

tags:

- mytag2

The template task in the above Ansible playbook takes two mandatory parameters src and dest. There are also a few optional parameters that can be specified in a template task but is not required at this stage.

* The src parameter specifies the name of the template file from templates directory that Ansible will copy to the remote server. In our case, the two templates files that we have created are index.html.j2 and nginx-vh.j2
* The dest parameter is the path in the remote server where the file should be placed.

Finally, run the playbook from your ansible working directory:

$ ansible-playbook playbook.yml -u USER

### Blocks

**Blocks**, which were introduced in version 2.0, allow you to logically group tasks and better handle errors, which is useful when you want to execute multiple tasks under a single condition.

To end the block, use the when keyword once you’re done defining all the tasks you want to be executed. If the evaluation of the when condition returns true, then all the tasks within the blocks will be executed one by one. All tasks within the blocks will inherit the common data or directives that you set just after the ‘when’ keyword.

---

- hosts: webservers

tasks:

- name: Install Nginx

block:

- apt: pkg=nginx state=present

- service: name=nginx state=started enabled=yes

when: ansible\_distribution == 'Ubuntu'

become: true

become\_user: root

The block section in the above playbook includes two related tasks to install nginx and start/enable it. The when evaluation specifies that these tasks should only be run when the remote managed host is using Ubuntu as its operating system. Both the tasks will inherit the privilege escalation directives after the ‘when’ keyword.

You can also use blocks to handle failures, similar to exceptions in most programming languages. The aim is to gracefully handle failures within the block rather than withdrawing the entire deployment.

Here is an example of how to use blocks to handle failures:

tasks:

- block:

- name: Enable Nginx during boot

service: name=nginx state=started enabled=yes

rescue:

- name: This section runs only when there is an error in the block.

debug: msg="There was an error in starting/enabling nginx."

always:

- name: This section will run always.

debug: msg="This always executes."`

### Conclusion:

Okay…so if you followed along with this Ansible tutorial step by step, you’ve gotten a simplified but practical lesson in most how to take your first steps with Ansible:

* How to install Ansible
* Running ad-hoc commands
* Understanding how modules work
* Creating Ansible playbooks
* Running your first playbook
* And a few other key fundamentals to get you started.
* This is by no means our final or definitive tutorial on Ansible—be sure to [subscribe to the Serverwise newsletter](https://ssdnodes.us18.list-manage.com/subscribe/post?u=7e134cc07e36ff0bacefca130&id=d1494787e6) to get more Ansible posts as they go live.

### Get more Ansible tutorial resources now:

In the meantime, you can check out these Ansible guides if you want to level up your knowledge immediately:

Check out our secure Ansible playbook tutorial to the complete playbook we put together for securing new VPSs:

* [Ansible playbook for a more secure VPS](https://blog.ssdnodes.com/blog/secure-ansible-playbook/) (part 1)
* [A More Secure Ansible Playbook](https://blog.ssdnodes.com/blog/secure-ansible-playbook-2/) (part 2)

Learn to us Ansible as a configuration management tool for setting up a bare CentOS, Debian, and Ubuntu server with more secure SSH settings. Plus a few tools to make your life a little easier:

* [Ansible tutorial: First steps with configuration management](https://blog.ssdnodes.com/blog/ansible-tutorial-getting-started/)

Thanks everyone…until next time!

[<img class="alignnone wp-image-4611 size-full" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020.png" alt="" width="1150" height="400" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020.png 1150w, https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020-300x104.png 300w, https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020-1024x356.png 1024w, https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020-768x267.png 768w" sizes="(max-width: 1150px) 100vw, 1150px" />](%3cimg%20class=%22alignnone%20wp-image-4611%20size-full%22%20src=%22https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020.png%22%20alt=%22%22%20width=%221150%22%20height=%22400%22%20srcset=%22https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020.png%201150w,%20https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020-300x104.png%20300w,%20https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020-1024x356.png%201024w,%20https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/End-of-post-2020-768x267.png%20768w%22%20sizes=%22(max-width:%201150px)%20100vw,%201150px%22%20/%3e)

[<img class="alignnone wp-image-4613 size-full" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/Blog-sidebar-2020-1.png" alt="" width="400" height="750" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/Blog-sidebar-2020-1.png 400w, https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/Blog-sidebar-2020-1-160x300.png 160w" sizes="(max-width: 400px) 100vw, 400px" />](%3cimg%20class=%22alignnone%20wp-image-4613%20size-full%22%20src=%22https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/Blog-sidebar-2020-1.png%22%20alt=%22%22%20width=%22400%22%20height=%22750%22%20srcset=%22https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/Blog-sidebar-2020-1.png%20400w,%20https://blog.ssdnodes.com/blog/wp-content/uploads/2020/03/Blog-sidebar-2020-1-160x300.png%20160w%22%20sizes=%22(max-width:%20400px)%20100vw,%20400px%22%20/%3e)

#### Share this post:

[Reddit](https://www.reddit.com/submit?url=https%3A%2F%2Fblog.ssdnodes.com%2Fblog%2Fstep-by-step-ansible-guide%2F&title=A+step+by+step+guide+to+Ansible+%28Tutorial%29) [Hacker News](http://news.ycombinator.com/submitlink?u=https%3A%2F%2Fblog.ssdnodes.com%2Fblog%2Fstep-by-step-ansible-guide%2F&t=A+step+by+step+guide+to+Ansible+%28Tutorial%29) [Facebook](http://www.facebook.com/sharer/sharer.php?s=100&p%5burl%5d=https%3A%2F%2Fblog.ssdnodes.com%2Fblog%2Fstep-by-step-ansible-guide%2F) [Twitter](https://twitter.com/intent/tweet?text=A+step+by+step+guide+to+Ansible+%28Tutorial%29+https%3A%2F%2Fblog.ssdnodes.com%2Fblog%2Fstep-by-step-ansible-guide%2F) [Flipboard](https://share.flipboard.com/bookmarklet/popout?v=A+step+by+step+guide+to+Ansible+%28Tutorial%29&url=https%3A%2F%2Fblog.ssdnodes.com%2Fblog%2Fstep-by-step-ansible-guide%2F)

Like what you saw? Subscribe to our weekly newsletter.

Top of Form

Subscribe



Bottom of Form

A note about tutorials: We encourage our users to try out tutorials, but they aren't fully supported by our team—we can't always provide support when things go wrong. Be sure to check which OS and version it was tested with before you proceed.

If you want a fully managed experience, with dedicated support for any application you might want to run, [contact us](https://www.ssdnodes.com/contact-us/) for more information.

**Tags:**[ansible](https://blog.ssdnodes.com/blog/tag/ansible/)[configuration management](https://blog.ssdnodes.com/blog/tag/configuration-management/)

### You might also like:

* [[](https://blog.ssdnodes.com/blog/ansible-tutorial-getting-started/)<img width="700" height="292" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/04/201704\_ansible-tutorial-700x292.jpeg" class="attachment-large size-large wp-post-image" alt="Ansible tutorial: Getting started with configuration management" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/04/201704\_ansible-tutorial-700x292.jpeg 700w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/04/201704\_ansible-tutorial-250x104.jpeg 250w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/04/201704\_ansible-tutorial-768x320.jpeg 768w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/04/201704\_ansible-tutorial-120x50.jpeg 120w" sizes="(max-width: 700px) 100vw, 700px" />](https://blog.ssdnodes.com/blog/ansible-tutorial-getting-started/)

[**Getting started with Ansible for configuration management (Tutorial)**](https://blog.ssdnodes.com/blog/ansible-tutorial-getting-started/)

* [[](https://blog.ssdnodes.com/blog/secure-ansible-playbook-2/)<img width="700" height="350" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170907\_ansible-v2-p2-700x350.png" class="attachment-large size-large wp-post-image" alt="" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170907\_ansible-v2-p2-700x350.png 700w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170907\_ansible-v2-p2-250x125.png 250w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170907\_ansible-v2-p2-768x384.png 768w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170907\_ansible-v2-p2-120x60.png 120w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170907\_ansible-v2-p2.png 1024w" sizes="(max-width: 700px) 100vw, 700px" />](https://blog.ssdnodes.com/blog/secure-ansible-playbook-2/)

[**Ansible security playbook for your VPS (part 2)**](https://blog.ssdnodes.com/blog/secure-ansible-playbook-2/)

* [[](https://blog.ssdnodes.com/blog/secure-ansible-playbook/)<img width="700" height="350" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170831\_ansible-secure-700x350.png" class="attachment-large size-large wp-post-image" alt="" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170831\_ansible-secure-700x350.png 700w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170831\_ansible-secure-250x125.png 250w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170831\_ansible-secure-768x384.png 768w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170831\_ansible-secure-120x60.png 120w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/09/20170831\_ansible-secure.png 1024w" sizes="(max-width: 700px) 100vw, 700px" />](https://blog.ssdnodes.com/blog/secure-ansible-playbook/)

[**Ansible security playbook for your VPS (part 1)**](https://blog.ssdnodes.com/blog/secure-ansible-playbook/)

* [[](https://blog.ssdnodes.com/blog/host-multiple-ssl-websites-docker-nginx/)<img width="700" height="292" src="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/11/201711\_docker-nginx-multiple-websites-700x292.jpeg" class="attachment-large size-large wp-post-image" alt="Docker Nginx multiple websites: a comprehensive tutorial" srcset="https://blog.ssdnodes.com/blog/wp-content/uploads/2017/11/201711\_docker-nginx-multiple-websites-700x292.jpeg 700w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/11/201711\_docker-nginx-multiple-websites-250x104.jpeg 250w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/11/201711\_docker-nginx-multiple-websites-768x320.jpeg 768w, https://blog.ssdnodes.com/blog/wp-content/uploads/2017/11/201711\_docker-nginx-multiple-websites-120x50.jpeg 120w" sizes="(max-width: 700px) 100vw, 700px" />](https://blog.ssdnodes.com/blog/host-multiple-ssl-websites-docker-nginx/)

[**Hosting multiple SSL-enabled sites with Docker and Nginx**](https://blog.ssdnodes.com/blog/host-multiple-ssl-websites-docker-nginx/)

PIPELINE SCRIPT FOR JENKINS ROLE OUT

# Creating your first Pipeline

Table of Contents

* [What is a Jenkins Pipeline?](https://jenkins.io/doc/pipeline/tour/hello-world/#what-is-a-jenkins-pipeline)
* [Quick Start Examples](https://jenkins.io/doc/pipeline/tour/hello-world/#examples)
  + [Java](https://jenkins.io/doc/pipeline/tour/hello-world/#java)
  + [Node.js / JavaScript](https://jenkins.io/doc/pipeline/tour/hello-world/#node-js-javascript)
  + [Ruby](https://jenkins.io/doc/pipeline/tour/hello-world/#ruby)
  + [Python](https://jenkins.io/doc/pipeline/tour/hello-world/#python)
  + [PHP](https://jenkins.io/doc/pipeline/tour/hello-world/#php)
  + [Go](https://jenkins.io/doc/pipeline/tour/hello-world/#go)

### What is a Jenkins Pipeline?

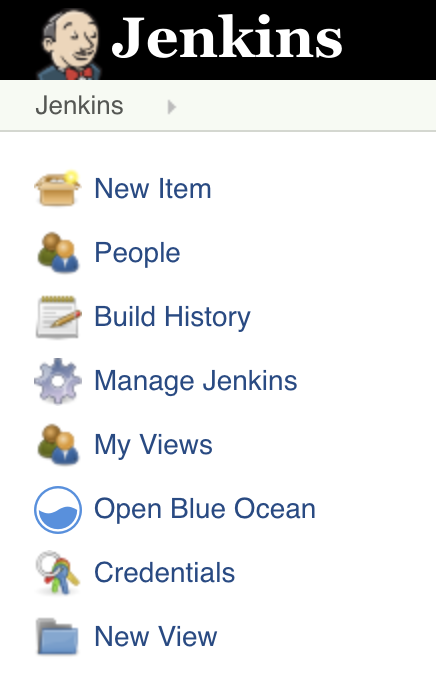
Jenkins Pipeline (or simply "Pipeline") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

A continuous delivery pipeline is an automated expression of your process for getting software from version control right through to your users and customers.

Jenkins Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code". The definition of a Jenkins Pipeline is typically written into a text file (called a Jenkinsfile) which in turn is checked into a project’s source control repository. [[1](https://jenkins.io/doc/pipeline/tour/hello-world/#_footnotedef_1)]

For more information about Pipeline and what a Jenkinsfile is, refer to the respective [Pipeline](https://jenkins.io/doc/book/pipeline) and [Using a Jenkinsfile](https://jenkins.io/doc/book/pipeline/jenkinsfile) sections of the User Handbook.

To get started quickly with Pipeline:

1. Copy one of the [examples below](https://jenkins.io/doc/pipeline/tour/hello-world/#examples) into your repository and name it Jenkinsfile
2. Click the **New Item** menu within Jenkins 
3. Provide a name for your new item (e.g. **My Pipeline**) and select **Multibranch Pipeline**
4. Click the **Add Source** button, choose the type of repository you want to use and fill in the details.
5. Click the **Save** button and watch your first Pipeline run!

You may need to modify one of the example Jenkinsfile's to make it run with your project. Try modifying the sh command to run the same command you would run on your local machine.

After you have setup your Pipeline, Jenkins will automatically detect any new Branches or Pull Requests that are created in your repository and start running Pipelines for them.

[**Continue to "Run multiple steps"**](https://jenkins.io/doc/pipeline/tour/running-multiple-steps)

## Quick Start Examples

Below are some easily copied and pasted examples of a simple Pipeline with various languages.

### Java

Jenkinsfile (Declarative Pipeline)

pipeline {

agent { docker { image 'maven:3.3.3' } }

stages {

stage('build') {

steps {

sh 'mvn --version'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world/) (Advanced)

Jenkinsfile (Scripted Pipeline)

/\* Requires the Docker Pipeline plugin \*/

node('docker') {

checkout scm

stage('Build') {

docker.image('maven:3.3.3').inside {

sh 'mvn --version'

}

}

}

### Node.js / JavaScript

Jenkinsfile (Declarative Pipeline)

pipeline {

agent { docker { image 'node:6.3' } }

stages {

stage('build') {

steps {

sh 'npm --version'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world/) (Advanced)

Jenkinsfile (Scripted Pipeline)

/\* Requires the Docker Pipeline plugin \*/

node('docker') {

checkout scm

stage('Build') {

docker.image('node:6.3').inside {

sh 'npm --version'

}

}

}

### Ruby

Jenkinsfile (Declarative Pipeline)

pipeline {

agent { docker { image 'ruby' } }

stages {

stage('build') {

steps {

sh 'ruby --version'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world/) (Advanced)

Jenkinsfile (Scripted Pipeline)

/\* Requires the Docker Pipeline plugin \*/

node('docker') {

checkout scm

stage('Build') {

docker.image('ruby').inside {

sh 'ruby --version'

}

}

}

### Python

Jenkinsfile (Declarative Pipeline)

pipeline {

agent { docker { image 'python:3.5.1' } }

stages {

stage('build') {

steps {

sh 'python --version'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world/) (Advanced)

Jenkinsfile (Scripted Pipeline)

/\* Requires the Docker Pipeline plugin \*/

node('docker') {

checkout scm

stage('Build') {

docker.image('python:3.5.1').inside {

sh 'python --version'

}

}

}

### PHP

Jenkinsfile (Declarative Pipeline)

pipeline {

agent { docker { image 'php' } }

stages {

stage('build') {

steps {

sh 'php --version'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world/) (Advanced)

Jenkinsfile (Scripted Pipeline)

/\* Requires the Docker Pipeline plugin \*/

node('docker') {

checkout scm

stage('Build') {

docker.image('php').inside {

sh 'php --version'

}

}

}

### Go

Jenkinsfile (Declarative Pipeline)

pipeline {

agent { docker { image 'golang' } }

stages {

stage('build') {

steps {

sh 'go version'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world/) (Advanced)

Jenkinsfile (Scripted Pipeline)

/\* Requires the Docker Pipeline plugin \*/

node('docker') {

checkout scm

stage('Build') {

docker.image('golang').inside {

sh 'go version'

}

}

}

[**Continue to "Running multiple steps"**](https://jenkins.io/doc/pipeline/tour/running-multiple-steps)

[Was this page helpful?](https://jenkins.io/doc/pipeline/tour/hello-world/#feedback)

Please submit your feedback about this page through this [quick form](https://jenkins.io/doc/feedback-form/).

Alternatively, if you don't wish to complete the quick form, you can simply indicate if you found this page helpful?

Top of Form



Yes     No

Type the answer to 5 plus 4 before clicking "Submit" below.  




Bottom of Form

See existing feedback [here](https://docs.google.com/spreadsheets/d/1IIdpVs39JDYKg0sLQIv-MNO928qcGApAIfdW5ohfD78).

[1](https://jenkins.io/doc/pipeline/tour/hello-world/#_footnoteref_1). [Source Control Management](https://en.wikipedia.org/wiki/Source_control_management)

* [Getting started](https://jenkins.io/doc/pipeline/tour/getting-started)
* [Creating your first Pipeline](https://jenkins.io/doc/pipeline/tour/hello-world)
* [Running multiple steps](https://jenkins.io/doc/pipeline/tour/running-multiple-steps)
* [Defining execution environments](https://jenkins.io/doc/pipeline/tour/agents)
* [Using environment variables](https://jenkins.io/doc/pipeline/tour/environment)
* [Recording test results and artifacts](https://jenkins.io/doc/pipeline/tour/tests-and-artifacts)
* [Cleaning up and notifications](https://jenkins.io/doc/pipeline/tour/post)
* [Deployment](https://jenkins.io/doc/pipeline/tour/deployment)

#### Tutorials

* [Overview](https://jenkins.io/doc/tutorials)
* [Build a Java app with Maven](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven)
* [Build a Node.js and React app with npm](https://jenkins.io/doc/tutorials/build-a-node-js-and-react-app-with-npm)
* [Build a Python app with PyInstaller](https://jenkins.io/doc/tutorials/build-a-python-app-with-pyinstaller)
* [Build a LabVIEW app](https://jenkins.io/doc/tutorials/build-a-labview-app)
* [Create a Pipeline in Blue Ocean](https://jenkins.io/doc/tutorials/create-a-pipeline-in-blue-ocean)
* [End-to-End Multibranch Pipeline Project Creation](https://jenkins.io/doc/tutorials/build-a-multibranch-pipeline-project)

#### Handbook

* [User Handbook overview](https://jenkins.io/doc/book/getting-started)
* [Installing Jenkins](https://jenkins.io/doc/book/installing)
* [Using Jenkins](https://jenkins.io/doc/book/using)
* [Pipeline](https://jenkins.io/doc/book/pipeline)
  + [Getting started with Pipeline](https://jenkins.io/doc/book/pipeline/getting-started)
  + [Using a Jenkinsfile](https://jenkins.io/doc/book/pipeline/jenkinsfile)
  + [Running Pipelines](https://jenkins.io/doc/book/pipeline/running-pipelines)
  + [Branches and Pull Requests](https://jenkins.io/doc/book/pipeline/multibranch)
  + [Using Docker with Pipeline](https://jenkins.io/doc/book/pipeline/docker)
  + [Extending with Shared Libraries](https://jenkins.io/doc/book/pipeline/shared-libraries)
  + [Pipeline Development Tools](https://jenkins.io/doc/book/pipeline/development)
  + [Pipeline Syntax](https://jenkins.io/doc/book/pipeline/syntax)
  + [Pipeline Best Practices](https://jenkins.io/doc/book/pipeline/pipeline-best-practices)
  + [Scaling Pipelines](https://jenkins.io/doc/book/pipeline/scaling-pipeline)
* [Blue Ocean](https://jenkins.io/doc/book/blueocean)
* [Managing Jenkins](https://jenkins.io/doc/book/managing)
* [System Administration](https://jenkins.io/doc/book/system-administration)
* [Scaling Jenkins](https://jenkins.io/doc/book/scaling)
* [Appendix](https://jenkins.io/doc/book/appendix)
* [Glossary](https://jenkins.io/doc/book/glossary)

#### Resources

* [Pipeline Syntax reference](https://jenkins.io/doc/book/pipeline/syntax)
* [Pipeline Steps reference](https://jenkins.io/doc/pipeline/steps)
* [LTS Upgrade Guide](https://jenkins.io/doc/upgrade-guide)

##### Tutorial Blog Posts

* [IntelliJ Setup for Jenkins Core Development](https://jenkins.io/doc/developer/building/intellij/)
* [Introducing Tutorials in the Jenkins User Documentation](https://jenkins.io/blog/2017/11/27/tutorials-in-the-jenkins-user-documentation/)
* [Pipeline Development Tools](https://jenkins.io/blog/2017/05/18/pipeline-dev-tools/)

[View all tutorial blog posts](https://jenkins.io/node/tags/tutorial)

[⇐ Using Jenkins](https://jenkins.io/doc/book/using)

[Index](https://jenkins.io/doc/book/)

[Getting started with Pipeline ⇒](https://jenkins.io/doc/book/pipeline/getting-started)

# Pipeline

Chapter Sub-Sections

* [Getting started with Pipeline](https://jenkins.io/doc/book/pipeline/getting-started)
* [Using a Jenkinsfile](https://jenkins.io/doc/book/pipeline/jenkinsfile)
* [Running Pipelines](https://jenkins.io/doc/book/pipeline/running-pipelines)
* [Branches and Pull Requests](https://jenkins.io/doc/book/pipeline/multibranch)
* [Using Docker with Pipeline](https://jenkins.io/doc/book/pipeline/docker)
* [Extending with Shared Libraries](https://jenkins.io/doc/book/pipeline/shared-libraries)
* [Pipeline Development Tools](https://jenkins.io/doc/book/pipeline/development)
* [Pipeline Syntax](https://jenkins.io/doc/book/pipeline/syntax)
* [Pipeline Best Practices](https://jenkins.io/doc/book/pipeline/pipeline-best-practices)
* [Scaling Pipelines](https://jenkins.io/doc/book/pipeline/scaling-pipeline)

Table of Contents

* [What is Jenkins Pipeline?](https://jenkins.io/doc/book/pipeline/#overview)
  + [Declarative versus Scripted Pipeline syntax](https://jenkins.io/doc/book/pipeline/#declarative-versus-scripted-pipeline-syntax)
* [Why Pipeline?](https://jenkins.io/doc/book/pipeline/#why)
* [Pipeline concepts](https://jenkins.io/doc/book/pipeline/#pipeline-concepts)
  + [Pipeline](https://jenkins.io/doc/book/pipeline/#pipeline)
  + [Node](https://jenkins.io/doc/book/pipeline/#node)
  + [Stage](https://jenkins.io/doc/book/pipeline/#stage)
  + [Step](https://jenkins.io/doc/book/pipeline/#step)
* [Pipeline syntax overview](https://jenkins.io/doc/book/pipeline/#pipeline-syntax-overview)
  + [Declarative Pipeline fundamentals](https://jenkins.io/doc/book/pipeline/#declarative-pipeline-fundamentals)
  + [Scripted Pipeline fundamentals](https://jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals)
* [Pipeline example](https://jenkins.io/doc/book/pipeline/#pipeline-example)

This chapter covers all recommended aspects of Jenkins Pipeline functionality, including how to:

* [get started with Pipeline](https://jenkins.io/doc/book/pipeline/getting-started) - covers how to [define a Jenkins Pipeline](https://jenkins.io/doc/book/pipeline/getting-started#defining-a-pipeline) (i.e. your Pipeline) through [Blue Ocean](https://jenkins.io/doc/book/pipeline/getting-started#through-blue-ocean), through the [classic UI](https://jenkins.io/doc/book/pipeline/getting-started#through-the-classic-ui) or in [SCM](https://jenkins.io/doc/book/pipeline/getting-started#defining-a-pipeline-in-scm),
* [create and use a Jenkinsfile](https://jenkins.io/doc/book/pipeline/jenkinsfile) - covers use-case scenarios on how to craft and construct your Jenkinsfile,
* work with [branches and pull requests](https://jenkins.io/doc/book/pipeline/multibranch),
* [use Docker with Pipeline](https://jenkins.io/doc/book/pipeline/docker) - covers how Jenkins can invoke Docker containers on agents/nodes (from a Jenkinsfile) to build your Pipeline projects,
* [extend Pipeline with shared libraries](https://jenkins.io/doc/book/pipeline/shared-libraries),
* use different [development tools](https://jenkins.io/doc/book/pipeline/development) to facilitate the creation of your Pipeline, and
* work with [Pipeline syntax](https://jenkins.io/doc/book/pipeline/syntax) - this page is a comprehensive reference of all Declarative Pipeline syntax.

For an overview of content in the Jenkins User Handbook, see [User Handbook overview](https://jenkins.io/doc/book/pipeline/getting-started).

## What is Jenkins Pipeline?

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the [Pipeline domain-specific language (DSL) syntax](https://jenkins.io/doc/book/pipeline/syntax). [[1](https://jenkins.io/doc/book/pipeline/#_footnotedef_1)]

The definition of a Jenkins Pipeline is written into a text file (called a [Jenkinsfile](https://jenkins.io/doc/book/pipeline/jenkinsfile)) which in turn can be committed to a project’s source control repository. [[2](https://jenkins.io/doc/book/pipeline/#_footnotedef_2)] This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

* Automatically creates a Pipeline build process for all branches and pull requests.
* Code review/iteration on the Pipeline (along with the remaining source code).
* Audit trail for the Pipeline.
* Single source of truth [[3](https://jenkins.io/doc/book/pipeline/#_footnotedef_3)] for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.

### Declarative versus Scripted Pipeline syntax

A Jenkinsfile can be written using two types of syntax - Declarative and Scripted.

Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

* provides richer syntactical features over Scripted Pipeline syntax, and
* is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline. Read more about how these two types of syntax differ in [Pipeline concepts](https://jenkins.io/doc/book/pipeline/#pipeline-concepts) and [Pipeline syntax overview](https://jenkins.io/doc/book/pipeline/#pipeline-syntax-overview) below.

## Why Pipeline?

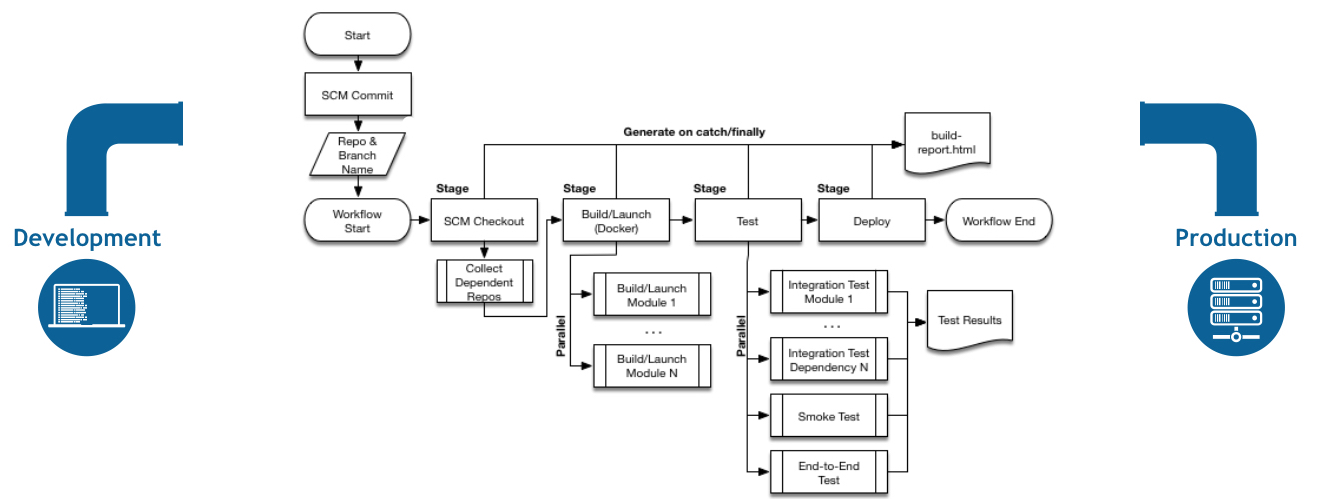
Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

* **Code**: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
* **Durable**: Pipelines can survive both planned and unplanned restarts of the Jenkins master.
* **Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
* **Versatile**: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
* **Extensible**: The Pipeline plugin supports custom extensions to its DSL [[1](https://jenkins.io/doc/book/pipeline/#_footnotedef_1)] and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, [[4](https://jenkins.io/doc/book/pipeline/#_footnotedef_4)] Pipeline makes this concept a first-class citizen in Jenkins.

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with [Pipeline Shared Libraries](https://jenkins.io/doc/book/pipeline/shared-libraries) and by plugin developers. [[5](https://jenkins.io/doc/book/pipeline/#_footnotedef_5)]

The flowchart below is an example of one CD scenario easily modeled in Jenkins Pipeline:



## Pipeline concepts

The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax (see the [overview](https://jenkins.io/doc/book/pipeline/#pipeline-syntax-overview) below).

### Pipeline

A Pipeline is a user-defined model of a CD pipeline. A Pipeline’s code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

Also, a pipeline block is a [key part of Declarative Pipeline syntax](https://jenkins.io/doc/book/pipeline/#declarative-pipeline-fundamentals).

### Node

A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.

Also, a node block is a [key part of Scripted Pipeline syntax](https://jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals).

### Stage

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress. [[6](https://jenkins.io/doc/book/pipeline/#_footnotedef_6)]

### Step

A single task. Fundamentally, a step tells Jenkins what to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh step: sh 'make'. When a plugin extends the Pipeline DSL, [[1](https://jenkins.io/doc/book/pipeline/#_footnotedef_1)] that typically means the plugin has implemented a new step.

## Pipeline syntax overview

The following Pipeline code skeletons illustrate the fundamental differences between [Declarative Pipeline syntax](https://jenkins.io/doc/book/pipeline/#declarative-pipeline-fundamentals) and [Scripted Pipeline syntax](https://jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals).

Be aware that both [stages](https://jenkins.io/doc/book/pipeline/#stage) and [steps](https://jenkins.io/doc/book/pipeline/#step) (above) are common elements of both Declarative and Scripted Pipeline syntax.

### Declarative Pipeline fundamentals

In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.

Jenkinsfile (Declarative Pipeline)

pipeline {

agent any **(1)**

stages {

stage('Build') { **(2)**

steps {

// **(3)**

}

}

stage('Test') { **(4)**

steps {

// **(5)**

}

}

stage('Deploy') { **(6)**

steps {

// **(7)**

}

}

}

}

|  |  |
| --- | --- |
| **1** | Execute this Pipeline or any of its stages, on any available agent. |
| **2** | Defines the "Build" stage. |
| **3** | Perform some steps related to the "Build" stage. |
| **4** | Defines the "Test" stage. |
| **5** | Perform some steps related to the "Test" stage. |
| **6** | Defines the "Deploy" stage. |
| **7** | Perform some steps related to the "Deploy" stage. |

### Scripted Pipeline fundamentals

In Scripted Pipeline syntax, one or more node blocks do the core work throughout the entire Pipeline. Although this is not a mandatory requirement of Scripted Pipeline syntax, confining your Pipeline’s work inside of a node block does two things:

1. Schedules the steps contained within the block to run by adding an item to the Jenkins queue. As soon as an executor is free on a node, the steps will run.
2. Creates a workspace (a directory specific to that particular Pipeline) where work can be done on files checked out from source control.  
   **Caution:** Depending on your Jenkins configuration, some workspaces may not get automatically cleaned up after a period of inactivity. See tickets and discussion linked from [JENKINS-2111](https://issues.jenkins-ci.org/browse/JENKINS-2111) for more information.

Jenkinsfile (Scripted Pipeline)

node { **(1)**

stage('Build') { **(2)**

// **(3)**

}

stage('Test') { **(4)**

// **(5)**

}

stage('Deploy') { **(6)**

// **(7)**

}

}

|  |  |
| --- | --- |
| **1** | Execute this Pipeline or any of its stages, on any available agent. |
| **2** | Defines the "Build" stage. stage blocks are optional in Scripted Pipeline syntax. However, implementing stage blocks in a Scripted Pipeline provides clearer visualization of each `stage’s subset of tasks/steps in the Jenkins UI. |
| **3** | Perform some steps related to the "Build" stage. |
| **4** | Defines the "Test" stage. |
| **5** | Perform some steps related to the "Test" stage. |
| **6** | Defines the "Deploy" stage. |
| **7** | Perform some steps related to the "Deploy" stage. |

## Pipeline example

Here is an example of a Jenkinsfile using Declarative Pipeline syntax - its Scripted syntax equivalent can be accessed by clicking the **Toggle Scripted Pipeline** link below:

Jenkinsfile (Declarative Pipeline)

pipeline { **(1)**

agent any **(2)**

options {

skipStagesAfterUnstable()

}

stages {

stage('Build') { **(3)**

steps { **(4)**

sh 'make' **(5)**

}

}

stage('Test'){

steps {

sh 'make check'

junit 'reports/\*\*/\*.xml' **(6)**

}

}

stage('Deploy') {

steps {

sh 'make publish'

}

}

}

}

[Toggle Scripted Pipeline](https://jenkins.io/doc/book/pipeline/) (Advanced)

Jenkinsfile (Scripted Pipeline)

node { **(7)**

stage('Build') { **(3)**

sh 'make' **(5)**

}

stage('Test') {

sh 'make check'

junit 'reports/\*\*/\*.xml' **(6)**

}

**if** (currentBuild.currentResult == 'SUCCESS') {

stage('Deploy') {

sh 'make publish'

}

}

}

|  |  |
| --- | --- |
| **1** | [pipeline](https://jenkins.io/doc/book/pipeline/syntax#declarative-pipeline) is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline. |
| **2** | [agent](https://jenkins.io/doc/book/pipeline/syntax#agent) is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline. |
| **3** | stage is a syntax block that describes a [stage of this Pipeline](https://jenkins.io/doc/book/pipeline/#stage). Read more about stage blocks in Declarative Pipeline syntax on the [Pipeline syntax](https://jenkins.io/doc/book/pipeline/syntax#stage) page. As mentioned [above](https://jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals), stage blocks are optional in Scripted Pipeline syntax. |
| **4** | [steps](https://jenkins.io/doc/book/pipeline/syntax#steps) is Declarative Pipeline-specific syntax that describes the steps to be run in this stage. |
| **5** | sh is a Pipeline [step](https://jenkins.io/doc/book/pipeline/syntax#steps) (provided by the [Pipeline: Nodes and Processes plugin](https://plugins.jenkins.io/workflow-durable-task-step)) that executes the given shell command. |
| **6** | junit is another a Pipeline [step](https://jenkins.io/doc/book/pipeline/syntax#steps) (provided by the [JUnit plugin](https://plugins.jenkins.io/junit)) for aggregating test reports. |
| **7** | node is Scripted Pipeline-specific syntax that instructs Jenkins to execute this Pipeline (and any stages contained within it), on any available agent/node. This is effectively equivalent to agent in Declarative Pipeline-specific syntax. |

Read more about Pipeline syntax on the [Pipeline Syntax](https://jenkins.io/doc/book/pipeline/syntax) page.

[1](https://jenkins.io/doc/book/pipeline/#_footnoteref_1). [Domain-specific language](https://en.wikipedia.org/wiki/Domain-specific_language)

[2](https://jenkins.io/doc/book/pipeline/#_footnoteref_2). [Source control management](https://en.wikipedia.org/wiki/Version_control)

[3](https://jenkins.io/doc/book/pipeline/#_footnoteref_3). [Single source of truth](https://en.wikipedia.org/wiki/Single_source_of_truth)

[4](https://jenkins.io/doc/book/pipeline/#_footnoteref_4). Additional plugins have been used to implement complex behaviors utilizing Freestyle Jobs such as the Copy Artifact, Parameterized Trigger, and Promoted Builds plugins

[5](https://jenkins.io/doc/book/pipeline/#_footnoteref_5). [GitHub Organization Folder plugin](https://plugins.jenkins.io/github-organization-folder)

[6](https://jenkins.io/doc/book/pipeline/#_footnoteref_6). [Blue Ocean](https://jenkins.io/doc/book/blueocean), [Pipeline: Stage View plugin](https://plugins.jenkins.io/pipeline-stage-view)

# Installing and Configuring Jenkins on Windows Server with IIS

by [Georg Dangl](https://blog.dangl.me/about/) in [Continuous Integration](https://blog.dangl.me/categories/Continuous%20Integration) Thursday, April 21, 2016

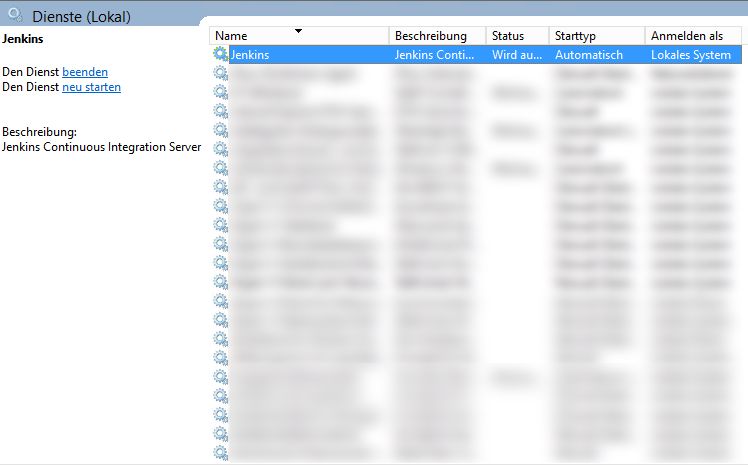
Thursday, April 21, 2016

Posted in [Jenkins](https://blog.dangl.me/tags/Jenkins) [IIS](https://blog.dangl.me/tags/IIS)

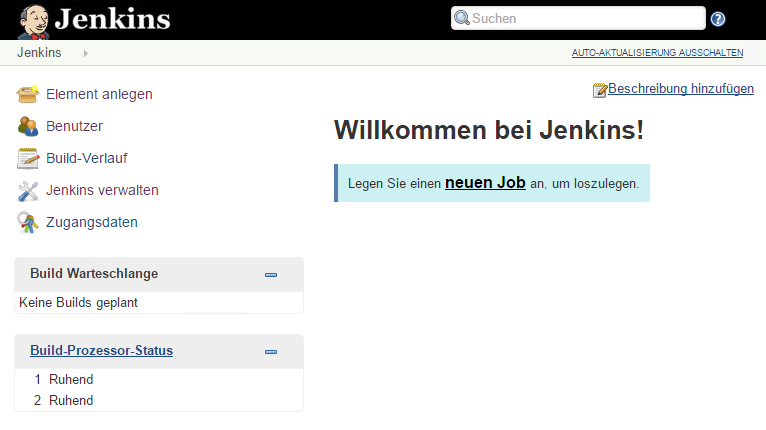
This post will give you a brief walkthrough of how to set up and configure a Jenkins Continuous Integration server on Windows, using IIS as forward proxy.

# Jenkins Installation

The actual installation is quite simple, just go to <https://jenkins.io>, download the latest Windows installer and perform the setup. You should take care to install Jenkins not under the default directory in C:\Program Files(x86)\Jenkins but to a different location, e.g. C:\Jenkins. After the installation, the "Jenkins" service should be present and set to automatic start.

[[](https://blog.dangl.me/media/1012/jenkinsservice_blurredd.jpg)](https://blog.dangl.me/media/1012/jenkinsservice_blurredd.jpg)

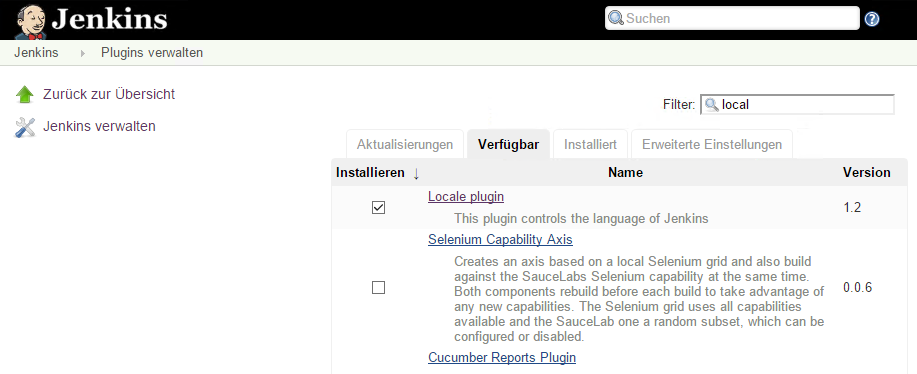
On completion, the setup process should open the default browser and redirect you to http://localhost:8080, the default Jenkins location. It might be that you're seeing a "Page not available" error due to Jenkins not having started up. In that case, just wait a few seconds and reload the page. You'll be greeted by your new butler, ready to serve.

[[](https://blog.dangl.me/media/1013/jenkinswelcome.png)](https://blog.dangl.me/media/1013/jenkinswelcome.png)

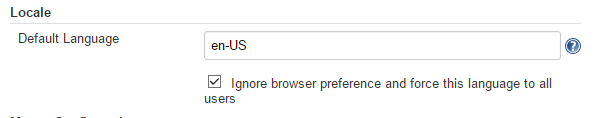
# Initial Jenkins Configuration

## Change Language to English

If your Jenkins locale is different from English you'll be greeted in that language, so instead of seeing "Welcome to Jenkins", you might have gotten " ". Since you probably want to change it to English, you're about to get familiar with Jenkins' plugin system right from the start. You have to navigate to Manage Jenkins -> Manage Plugins -> Available (or the equivalent in your language), search for the "Locale Plugin" and install it.

[[](https://blog.dangl.me/media/1014/jenkinslocaleplugin.png)](https://blog.dangl.me/media/1014/jenkinslocaleplugin.png)

After you've installed the plugin and restarted Jenkins, there's a new section in the Manage Jenkins -> Configure System settings menu called Locale where you need to apply the following settings:

[[](https://blog.dangl.me/media/1015/jenkinslocalesettings.png)](https://blog.dangl.me/media/1015/jenkinslocalesettings.png)

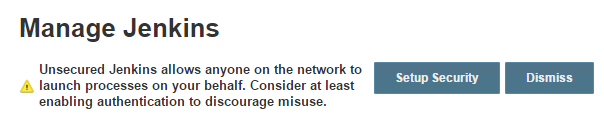
Congratulations, you just installed and used your first plugin. !

## Configure Email

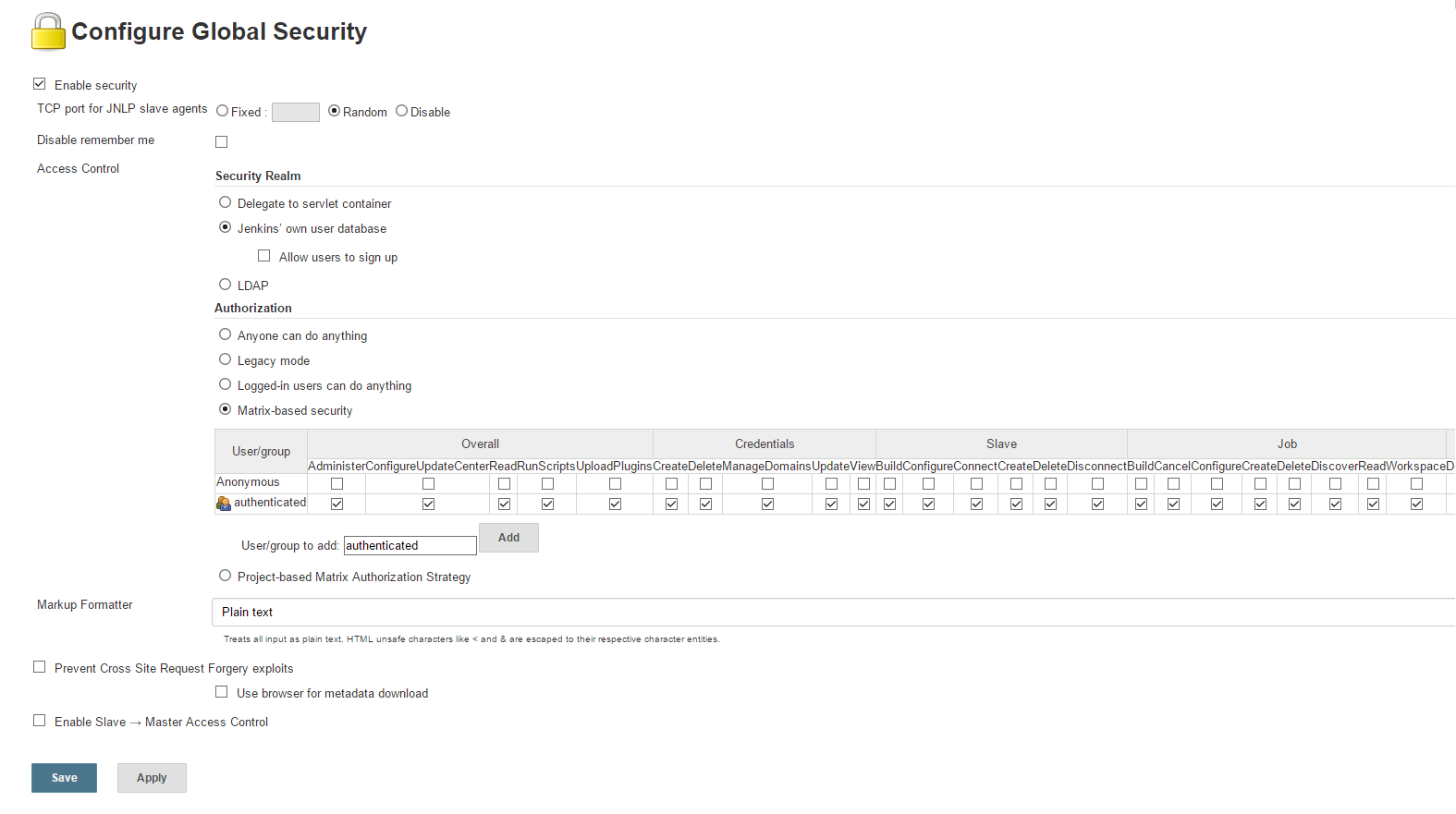
Configuring email settings is pretty straightforward. Just go to Manage Jenkins -> Configure System and scroll down to the email settings. You'll need to enter the credentials of the email account you want to use and you should verify the setup by sending a test email. Just remember to set the "System Admin e-mail address" field so Jenkins uses the correct sender field when sending emails and your sent mails don't get rejected by the SMTP server.

## Securing Jenkins

When you're in the Manage Jenkins section, you'll notice that it's not so hard to miss the hint for taking security into consideration.

[[](https://blog.dangl.me/media/1016/jenkinssecurityreminder.png)](https://blog.dangl.me/media/1016/jenkinssecurityreminder.png)

After clicking on "Setup Security", you should set up the following configuration:

[[](https://blog.dangl.me/media/1017/jenkinssecurityconfig.png)](https://blog.dangl.me/media/1017/jenkinssecurityconfig.png)

It's obvious that you want to enable security. To start quickly, you can tell Jenkins to use and maintain its own user database. Disallow users to sign up, since you want manual control over who can access the server. In the "Authorization" section, you should select "Matrix-based security" and then add the group "authenticated" to the settings. Enable all checkboxes for them and none for the "Anonymous" group. Now when you click save and navigate to another page, you'll be asked to create a user (since there is none yet, and you require one user, of course!). More sophisticated user rights management is not part of this tutorial. If you have a need for that, you can dig into the official documentation when you've finished setting up the basics!

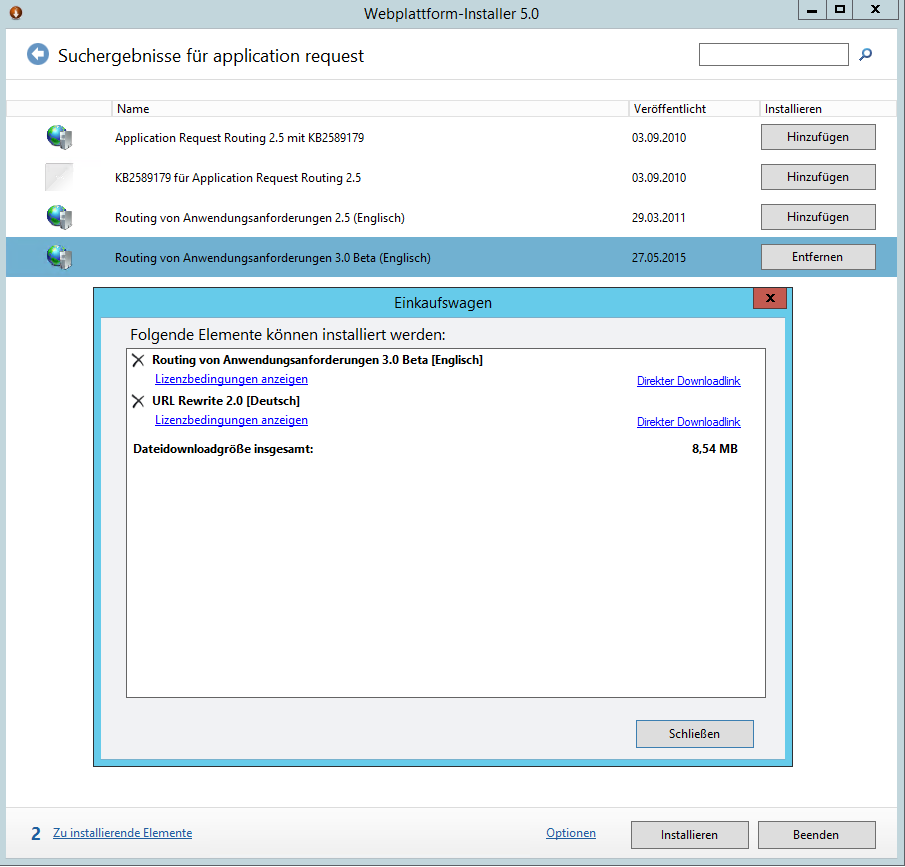
Adding a user can easily be done in the Manage Jenkins menu by going to Manage Users and then select Create User in the left navigation menu.

## Setting up an IIS Reverse Proxy to Access Jenkins from the Outside

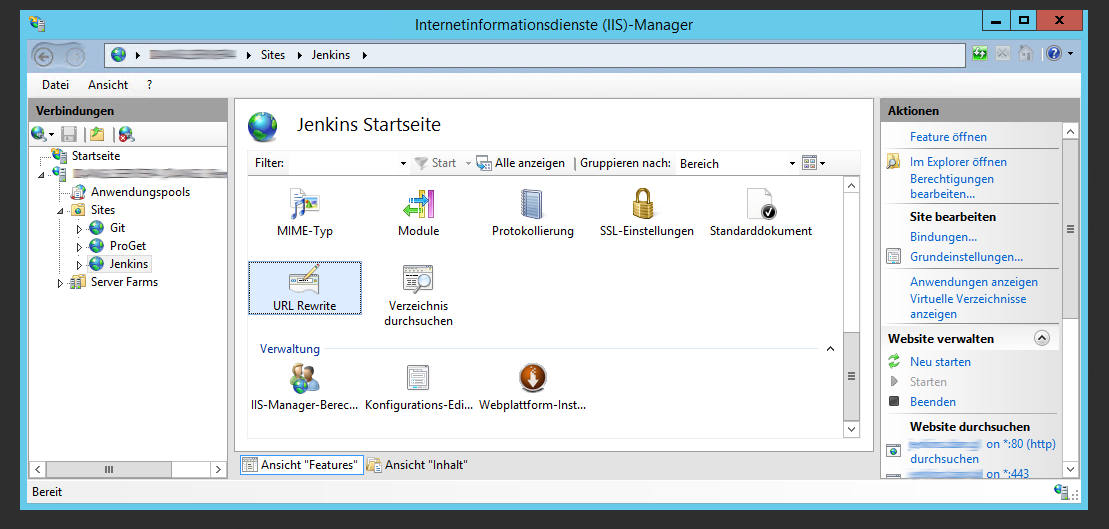
Now that Jenkins is secured, it's time to make it accessible from outside of the local machine without having to open ports and accessing Jenkins directly. Bonus: If you're planning to use SSL (which you should!) it's much easier to do that using IIS instead of the Java tools for that if you're more experienced in the Microsoft world than the Java one.

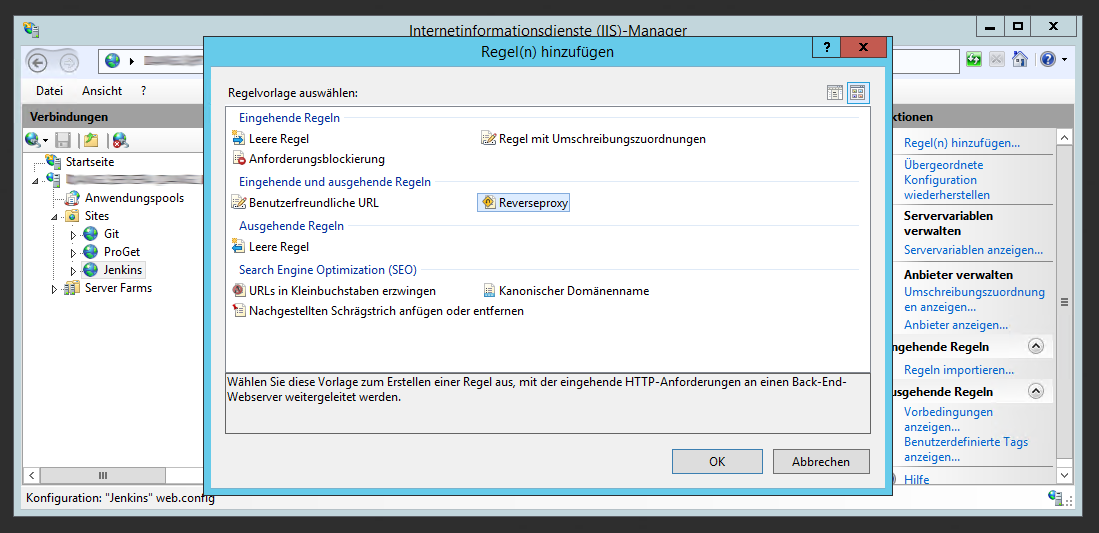
First, we need the correct IIS plugins, available in the WebPlatform-Installer. I hadn't set my OS language to English, so you'll have to rely on my translations enough to search for the required components.

1. Application Request Routing
2. URL Rewrite

[[](https://blog.dangl.me/media/1018/iis-url-reqrite-arr.png)](https://blog.dangl.me/media/1018/iis-url-reqrite-arr.png)

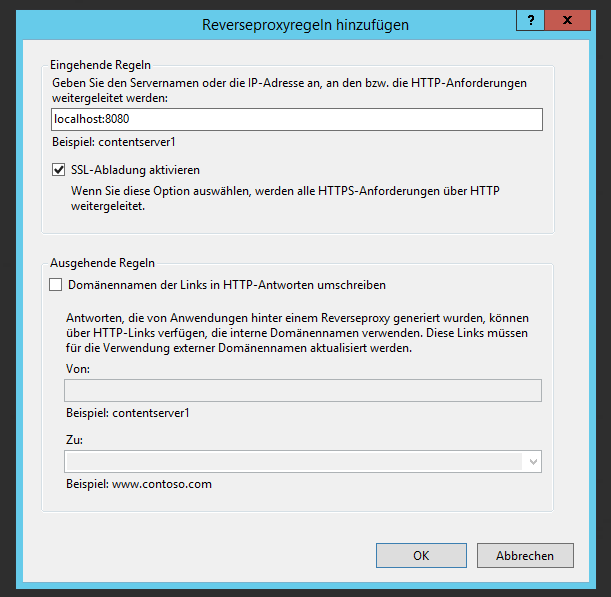
After you've installed them, just create a new website in IIS and assign it the bindings that you want to use for accessing Jenkins, like the preferred IP addresses to listen on and the hostnames to bind to.  
The next step is to select the site, go to "URL Rewrite" and create a new rule by selecting "Add Rule" on the right action menu.

[[](https://blog.dangl.me/media/1019/selecturlrewrite_blurred.png)](https://blog.dangl.me/media/1019/selecturlrewrite_blurred.png)

[[](https://blog.dangl.me/media/1020/urlrewritecreatereverseproxy_blurred.png)](https://blog.dangl.me/media/1020/urlrewritecreatereverseproxy_blurred.png)

The first time you're creating a new reverse proxy rule, IIS Manager will ask you to add reverse proxy rules. You can confirm that, since you do want it because you just clicked on that=)

Now, let's examine the necessary configuration:

[[](https://blog.dangl.me/media/1021/reverseproxyruleconfiguration.png)](https://blog.dangl.me/media/1021/reverseproxyruleconfiguration.png)

* The address to which the request should be routed will be the local Jenkins URL, it's the one we've used so far to access Jenkins.
* SSL Offload: Activate it when Jenkins has not been configured for SSL. You'll want to use SSL for the access from the outside, but the internal communication from IIS to Jenkins may very well happen unencrypted. It should be disabled if your IIS needs to route over some untrusted network path to talk to your Jenkins (so when Jenkins and IIS are neither in the same network nor on the same machine).
* Domain name rewriting is in practice not too different from how you'd expect string.Replace() would work it's replacing the internal domain with the external one. Since Jenkins does have a feature for that you shouldn't enable it here, if only for performance reasons.

Click OK and then navigate to where your IIS site is listening on. If everything worked, you should see Jenkins!

If you navigate to the directory that you created for your reverse proxy website, you'll find that a web.config containing a rewrite rule has been created. If you're using SSL, you might alter it so that the "rules" element contains these two rules:

[?](https://blog.dangl.me/archive/installing-and-configuring-jenkins-on-windows-server-with-iis/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!-- Enforces redirection of all HTTP traffic to HTTPS -->  <rule name="Enforce HTTPS" stopProcessing="true">      <match url="(.\*)" />          <conditions>              <add input="{HTTPS}" pattern="off" />          </conditions>      <action type="Redirect" url="https://{HTTP\_HOST}/{R:1}" redirectType="Permanent" />  </rule>  <rule name="ReverseProxyToLocalJenkinsRule" stopProcessing="true">      <match url="(.\*)" />          <conditions>              <add input="{HTTPS}" pattern="on" />          </conditions>          <action type="Rewrite" url="<http://localhost:8080/>{R:1}" />  </rule> |

These rules will return a permanent redirect result to your secure endpoint for any requests if they're being sent over plain http. . The second rule is the original reverse proxy rule, now altered to only trigger when the request is coming via https. If you're not using SSL, then you should really start using it~~, there's free certificates for you available at~~ [~~StartSSL~~](https://www.startssl.com).

**Update 30.01.2017:** [With StartSSL being effectively dead](https://linustechtips.com/main/topic/688200-apple-google-and-mozilla-disavow-wosign-and-startcom-certificates/), they're no longer an option for free TLS certificates. However, [Let's Encrypt](https://letsencrypt.org) being supported by all major players now, they're a great source of free TLS certificates for web servers. You can [read here](https://blog.dangl.me/archive/grabbing-a-lets-encrypt-certificate-with-umbraco/) how to obtain certificates in IIS with Let's Encrypt.

Now, but this is optional, you might get 404.11 URL\_DOUBLE\_ESCAPED errors for certain pages in your Jenkins application from IIS. You can add the following section directly within the system.webServer section of the web.config file to prevent this error.

[?](https://blog.dangl.me/archive/installing-and-configuring-jenkins-on-windows-server-with-iis/)

|  |  |
| --- | --- |
| 1  2  3 | <security>      <requestFiltering allowDoubleEscaping="true" />  </security> |

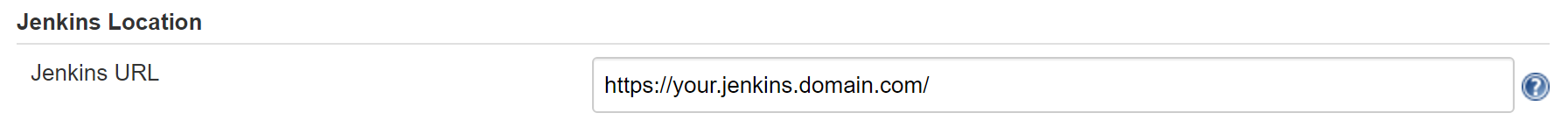
Finally, depending on your unit test namespaces and classnames, you might run into an issue where the urls generated by Jenkins are too long for IIS. Just add the following snippet to your web.config file and you should be save for a while. Please note that certain browsers might impose their own limitations here, but you're unlikely to get urls so long as to cause issues there.

[?](https://blog.dangl.me/archive/installing-and-configuring-jenkins-on-windows-server-with-iis/)

|  |  |
| --- | --- |
| 1  2  3 | <system.web>      <httpRuntime maxUrlLength="4096"/>  </system.web> |

 After the IIS configuration, two more things need to be done in Jenkins:

* Tell Jenkins the domain it will be available on. It's to let Jenkins know what its "real" address should be. Do that in "Manage Jenkins" -> "Configure System" -> "Jenkins Location" by replacing the Jenkins URL with the one to actually use, e.g. jenkins.yourdomain.com.  
  Don't do this before you've set up the reverse proxy, since for me I wasn't able to log in anymore using the localhost:8080 address after I logged out. Might been some weird thing on my machine, but if you do happen to lock yourself out you're going to have to disable global security in the config.xml in Jenkins install directory so you can recover.

[[](https://blog.dangl.me/media/1022/setjenkinslocation.png)](https://blog.dangl.me/media/1022/setjenkinslocation.png)

(After that, you'll get a notification in the Manage Jenkins section telling you that your reverse proxy configuration is broken. But what's not there yet can't be broken, so don't worry)

* Configure Jenkins so that it will only serve requests on localhost. The firewall should prevent such a thing from happening, but it's a good idea to configure it anyway. The Windows Firewall is kinda prone to being toyed with when stuff stops working and then it's just never set back to how it should be.  
  Note: Don't do this if your webserver will be on another host. In that case, you do want to access it from somewhere other than the local machine, meaning that requests will come from some external facing IP address. If that's the case, it's best to set up a whitelist to allow access on that port.  
  As of now, you can only change that in the jenkins.xml file that is located in the Jenkins install folder. Just open it, search the element "arguments" and add the following argument: --httpListenAddress=127.0.0.1  
  We're telling Jenkins to only listen on 127.0.0.1, instead of the default which makes it to listen to anything (like I do with music). Restart the service to have the changes take effect.

Now we've got a basic setup for Jenkins. It doesn't do much yet, but I'll write some more posts soon telling you how to setup .Net projects (regular and with the new dotnet CLI) and some other stuff.

[Next: Install a Windows Server Active Directory Root Certificate in iOs](https://blog.dangl.me/archive/install-a-windows-server-active-directory-root-certificate-in-ios/)

#### Share this post

All Courses

Top of Form

Bottom of Form

[Log in](https://lms.simplilearn.com)

[Resources](https://www.simplilearn.com/resources) Virtualization and Cloud Computing

[Agile and Scrum](https://www.simplilearn.com/resources/agile-and-scrum)[Big Data and Analytics](https://www.simplilearn.com/resources/big-data-and-analytics)[Digital Marketing](https://www.simplilearn.com/resources/digital-marketing)[IT Security Management](https://www.simplilearn.com/resources/it-security-management)[IT Service and Architecture](https://www.simplilearn.com/resources/it-service-management)[Project Management](https://www.simplilearn.com/resources/project-management)[Salesforce Training](https://www.simplilearn.com/resources/salesforce-training)[Virtualization and Cloud Computing](https://www.simplilearn.com/resources/virtualization-and-cloud-computing)[Career Fast-track](https://www.simplilearn.com/resources/career-fast-track)[Enterprise](https://www.simplilearn.com/resources/enterprise)[Digital Transformation](https://www.simplilearn.com/resources/digital-transformation)[Other Segments](https://www.simplilearn.com/resources/other-segments)

[Articles](https://www.simplilearn.com/resources/virtualization-and-cloud-computing/articles)[Ebooks](https://www.simplilearn.com/resources/virtualization-and-cloud-computing/ebooks?source=frs_header)[On-demand Webinars](https://www.simplilearn.com/resources/virtualization-and-cloud-computing/on-demand-webinars?source=frs_header)

More

[Free Practice Tests](https://www.simplilearn.com/resources/virtualization-and-cloud-computing/free-practice-tests)

#### Trending now

### [Git Push Command Explained](https://www.simplilearn.com/git-push-command-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/git-push-command-article?source=frs_left_nav_clicked" \t "_self)

### [The Basics of Docker Swarm](https://www.simplilearn.com/basics-of-docker-swarm-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/basics-of-docker-swarm-article?source=frs_left_nav_clicked" \t "_self)

### [Cloud Migration: Tips to Complete a Successful Migration to AWS and Microsoft Azure](https://www.simplilearn.com/cloud-migration-to-aws-and-azure-guide-pdf?source=frs_left_nav_clicked" \t "_self)

[Ebook](https://www.simplilearn.com/cloud-migration-to-aws-and-azure-guide-pdf?source=frs_left_nav_clicked" \t "_self)

### [Cloud Computing Architecture](https://www.simplilearn.com/cloud-computing-architecture-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/cloud-computing-architecture-article?source=frs_left_nav_clicked" \t "_self)

### [Top 93 AWS Interview Questions and Answers to Know Before Your Next Interview](https://www.simplilearn.com/aws-interview-questions-and-answers-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/aws-interview-questions-and-answers-article?source=frs_left_nav_clicked" \t "_self)

### [10 years of Development Methodologies](https://www.simplilearn.com/ten-years-of-development-methodologies-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/ten-years-of-development-methodologies-article?source=frs_left_nav_clicked" \t "_self)

### [AWS vs. Azure: Choosing The Right Cloud Platform](https://www.simplilearn.com/aws-vs-azure-cloud-certification-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/aws-vs-azure-cloud-certification-article?source=frs_left_nav_clicked" \t "_self)

### [Citrix Xenserver Vs Vmware vSphere](https://www.simplilearn.com/citrix-xenserver-vs-vmware-vsphere-rar338-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/citrix-xenserver-vs-vmware-vsphere-rar338-article?source=frs_left_nav_clicked" \t "_self)

### [The Top Cloud Computing Skills You Need To Pick Up This Year](https://www.simplilearn.com/top-cloud-computing-skills-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/top-cloud-computing-skills-article?source=frs_left_nav_clicked" \t "_self)

### [6 Reasons a Microsoft Azure Certification is the Secret to a Great Cloud Computing Career](https://www.simplilearn.com/reasons-microsoft-azure-certification-secret-to-cloud-computing-career-article?source=frs_left_nav_clicked" \t "_self)

[Article](https://www.simplilearn.com/reasons-microsoft-azure-certification-secret-to-cloud-computing-career-article?source=frs_left_nav_clicked" \t "_self)