**Learning Outcomes**

After completing this Elective module a student will be able to:

1. IBM Cloud DevOps Toolchain & Services

# IBM Cloud DevOps

Delivering software and services at the speed the market demands requires teams to iterate and experiment rapidly. They must deploy new versions frequently, driven by feedback and data. The most successful cloud development teams adopt modern DevSecOps culture and practices, embrace cloud-native architectures, and assemble toolchains from best-in-class tools to unleash their productivity. Doing these things quickly is a key competitive advantage.

The [IBM Cloud Garage Method](https://www.ibm.com/cloud/garage) describes architectures, practices, and DevOps toolchains to allow enterprises to innovate at scale. Use the IBM Cloud Garage Method to help you to transform your culture and use the tools effectively and fast.

An enterprise application Developer can start building and deploying cloud-native applications in minutes. They can use a full set of services to build cognitive, IoT, blockchain, mobile, and data-intensive applications. With the IBM Cloud App Service, an individual Developer can create a project, pick an application starter kit, and deploy a production-ready application to IBM Cloud. The platform's code generation technology creates a starter application in the Developer's preferred language and framework, which is tailored to their needs and use case. Any services that are required in support of the use case, such as Watson Conversation, are provisioned automatically. Developers can debug and test on their local workstation or in the cloud, and use a DevOps toolchain to collaborate with others and automate the delivery process.

As team members join a project, they need an integrated set of tools that span development, deployment, and production operations. IBM's Open Toolchain architecture enables a team to rapidly provision best-in-class DevSecOps tools from IBM, open source, and others. Integrations between these tools are configured automatically. Toolchains are a first class concept on the platform, so Developers can quickly organize everything that they need in one place, and evolve the toolchain over time. IBM provides toolchain templates that support Garage Method best practices, which you can customize to promote proven toolchain patterns across your enterprise.

IBM Cloud® Continuous Delivery provides a core set of tools for any DevSecOps toolchain: Git Repos and Issue Tracking, Delivery Pipeline, and Eclipse Orion Web IDE. Git Repos and Issue Tracking is based on the GitLab Community Edition, and offers planning boards and source code collaboration through merge requests. The Delivery Pipeline orchestrates build, test, and deployment jobs across multiple environments as changes progress from the Developer to production. Applications can be deployed in minutes to the Cloud Foundry environment or to a Kubernetes cluster on IBM Cloud, to either public or private clouds. The Eclipse Orion Web IDE gives Developers quick access to the code from any browser.

Open toolchain integrates more tools around Continuous Delivery such as Slack, Atlassian JIRA, Sonatype Nexus, JFrog Artifactory, Sauce Labs, PagerDuty, IBM Cloud Availability Monitoring, IBM Vulnerability Advisor, and IBM Globalization Pipeline. You can also substitute other tools for the Continuous Delivery capabilities, including GitHub and Jenkins. Developers can also use their favorite IDEs and editors, such as Visual Studio Code, Eclipse, and more.

Code repos, issue tracking systems, build systems, and deployment systems represent a wealth of data that can be used to help you deliver apps more efficiently and effectively. IBM Cloud® DevOps Insights uses big data analysis to provide valuable insights to Executives, Managers, and Developers. DevOps Insights aggregates and analyzes data from your DevOps toolchain to advise you about the risk of deploying specific changes, and areas to improve both your codebase and team productivity. The Delivery Pipeline can automatically gate deployment to an environment based on the risk of a change.

IBM Cloud DevOps provides concrete practices and architectures for cloud development. It enables Developers to get started quickly with new projects that employ the rich catalog of services on the IBM Cloud. IBM Cloud DevOps also provides Developers an open and integrated set of tools for automating delivery with speed and control.

# DevOps toolchains

A DevOps toolchain is a set of tools that automates the tasks of developing and deploying your app. You can perform DevOps manually with simple apps, but the need for automation increases quickly as app complexity increases, and toolchain automation is a must-have for continuous delivery.

The core component of a DevOps toolchain is a version control repository like GitHub. More tools might include backlog tracking, delivery pipelines, an integrated development environment (IDE), and monitoring like IBM Cloud® DevOps Insights.

When you [create an app](https://cloud.ibm.com/docs/apps?topic=apps-getting-started) by using a starter kit, and then click **Deploy my app** on the App details page, a DevOps toolchain is created. The toolchain has a code repository, delivery pipeline, and web IDE. You can then build on this toolchain to collaboratively manage and deploy your app to separate environments for development, test, and production.

## Toolchain templates

You can use a template as a starting point to [create a toolchain](https://cloud.ibm.com/devops/create). Toolchain templates include specific sets of tool integrations that support development, deployment, and operations tasks.

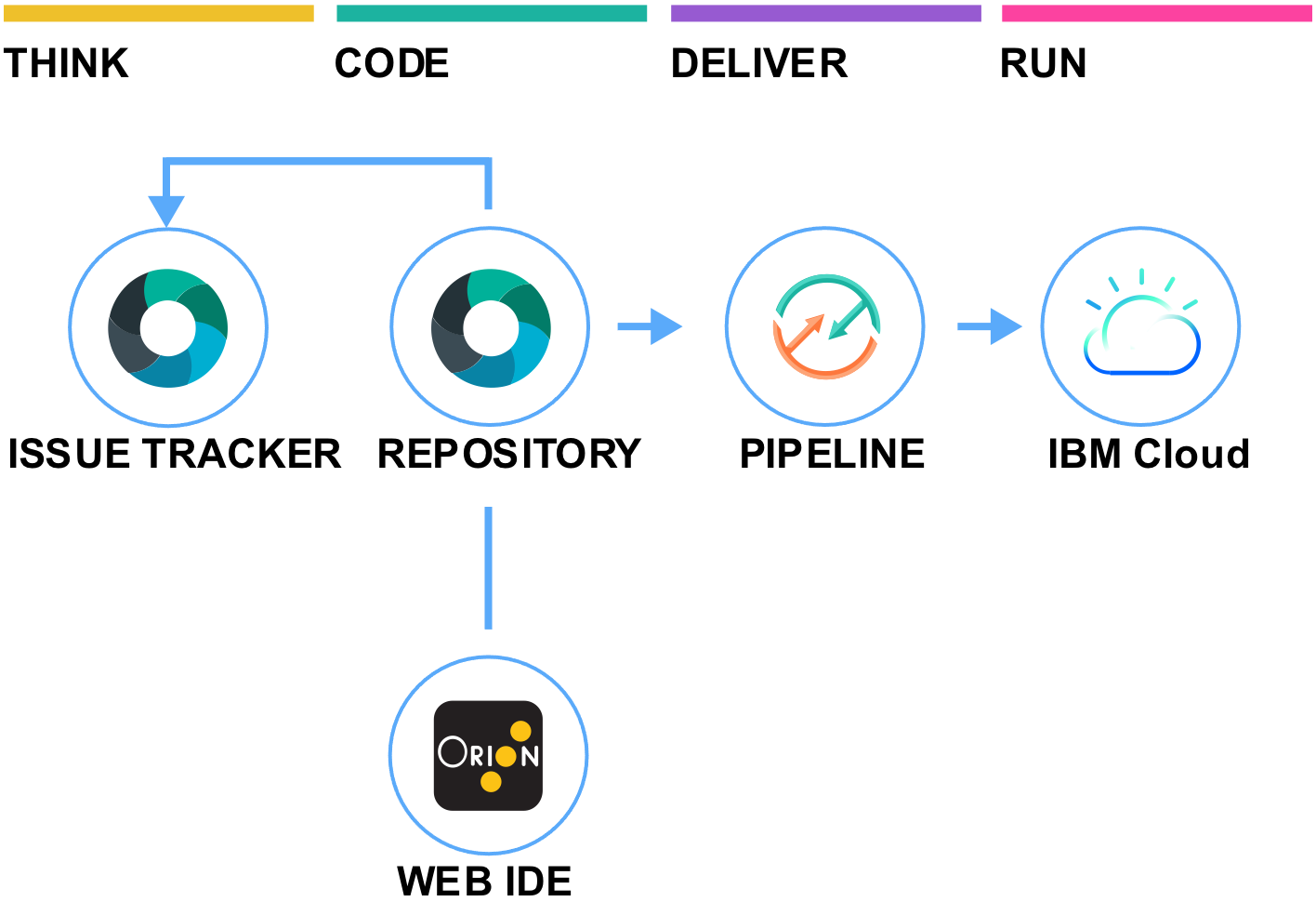
Some toolchain templates include tool integrations that are part of the Continuous Delivery service. If an instance of that service isn't already in your resource group, when you click **Create** to create the toolchain, the service is automatically added with the selected free Lite plan. For more information and terms, see the [IBM Cloud catalog](https://cloud.ibm.com/catalog/services/continuous-delivery/).

The "Develop and test microservices on Cloud Foundry" toolchain deploys an app with catalog and orders APIs that are backed by a Cloudant store. As part of deploying the app, a no-cost Cloudant service instance is created. For more information and terms, see the [IBM Cloud catalog](https://cloud.ibm.com/catalog/services/cloudant-nosql-db/).

The predefined DevOps toolchain templates are recommended examples that solve real world scenarios and each contains a sample app. You can use your own app by specifying your Git repo when you create the toolchain from the template.

| **Template and Available Regions** | **Description and Available Tutorials** | **Included Tools** |
| --- | --- | --- |
| [“Develop a Cloud Foundry app” toolchain](http://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fsimple-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can develop and deploy a Cloud Foundry app. By default, this toolchain uses a sample Node.js "Hello world" app, but you can link to your own GitHub repo instead. The toolchain is preconfigured for continuous delivery, source control, issue tracking, and online editing.  Try the tutorial: [Introduce Toolchains by using the “Develop a Cloud Foundry app” toolchain](https://www.ibm.com/cloud/garage/tutorials/introduce-develop-cloud-foundry-app-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  GitHub and Issues  IBM Cloud |
| [“Develop a Code Engine app" toolchain](http://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fcode-engine-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can develop and deploy an application securely into Code Engine, a shared, multi-tenant Kubernetes service on IBM Cloud. By default, the toolchain uses a sample Node.js "Hello World" app, but you can link to your own GitHub repository instead. This toolchain is preconfigured for continuous delivery, source control, issue tracking, and online editing.  Try the tutorial: [Develop and deploy an app by using Code Engine](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-tutorial-cd-code-engine) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud® Code Engine |
| [“Develop a Kubernetes app" toolchain](http://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fsecure-kube-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can develop, and deploy an application securely into a Kubernetes cluster that is managed by the IBM Cloud Kubernetes Service. By default, the toolchain uses a sample Node.js "Hello World" app, but you can link to your own GitHub repository instead. This toolchain is preconfigured for continuous delivery with Vulnerability Advisor, source control, issue tracking, and online editing.  Try the tutorial: [Use the "Develop a Kubernetes app" toolchain](https://www.ibm.com/cloud/garage/tutorials/use-develop-kubernetes-app-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  GitHub and Issues  IBM Cloud Kubernetes Service (Kubernetes cluster) |
| [“Develop a Kubernetes app with Razee" toolchain](http://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fkube-razee-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can develop an application, and deploy changes by using a Razee agent in your Kubernetes cluster. By default, the toolchain uses a sample Node.js "Hello World" app, but you can link to your own GitHub repository instead. This toolchain is preconfigured for continuous delivery with Vulnerability Advisor, source control, issue tracking, and online editing.  Try the tutorial: [Use the "Develop a Kubernetes app with Razee" toolchain](https://www.ibm.com/cloud/garage/tutorials/use-develop-kubernetes-app-with-razee-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud Kubernetes Service (Kubernetes cluster) |
| [“Develop a Kubernetes app with Helm" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fsimple-helm-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can develop a Docker application and its Helm chart together in source control and build and deploy it automatically to a Kubernetes cluster. The toolchain performs smoke tests before building or deploying your app and ensures privacy by using a private container registry and namespaces for the container registry and the Kubernetes cluster. This toolchain also uses Vulnerability Advisor to ensure that only secure images get deployed.  Try the tutorial: [Use the "Develop a Kubernetes app with Helm" toolchain](https://www.ibm.com/cloud/garage/tutorials/use-develop-kubernetes-app-with-helm-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud Kubernetes Service (Kubernetes cluster) with a Helm chart |
| ["Develop and test microservices on Kubernetes with Helm" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fmicroservices-helm-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this cloud-native toolchain, you can use a combination of continuous integration and continuous deployment pipelines to coordinate individually developed microservices into releases that are promoted across environments. This toolchain uses a sample online store app that consists of three microservices: a Catalog API, an Orders API, and a user interface that calls both of these APIs. The toolchain is preconfigured for continuous delivery, source control, functional testing, issue tracking, online editing, and alert notification.  Try the tutorial: [Use the "Develop and test microservices with Kubernetes and Helm" toolchain](https://www.ibm.com/cloud/garage/tutorials/use-develop-test-microservices-with-kubernetes-and-helm-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud® DevOps Insights  PagerDuty  Sauce Labs  IBM Cloud Kubernetes Service (Kubernetes cluster) with a Helm chart |
| ["Develop a Knative service app" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fknative-service-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this Knative toolchain, you can continuously deliver a secure Docker app to a Kubernetes cluster by using Knative. This toolchain is preconfigured for continuous delivery with Vulnerability Advisor, source control, issue tracking, online editing, and Knative deployment to the IBM Cloud Kubernetes Service. | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking |
| ["Canary testing in Kubernetes using Istio" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fcanary-testing-istio-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can securely develop, A/B test, and deploy an app into a Kubernetes cluster that is managed by the IBM Cloud Kubernetes Service. Although the toolchain uses a sample Node.js Hello World app by default, you can link to your own GitHub repo instead. This toolchain is preconfigured for continuous delivery with A/B testing, Vulnerability Advisor, source control, issue tracking, and online editing.  Try the tutorial: [Use the "Run your first canary test in Kubernetes using the Istio toolchain"](https://www.ibm.com/cloud/garage/tutorials/use-canary-testing-in-kubernetes-using-istio-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud Kubernetes Service (Kubernetes cluster) |
| ["Progressive rollout in Kubernetes using iter8" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fiter8-toolchain-rollout)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can securely develop, build, and roll out an application into a Kubernetes cluster that is managed by the IBM Cloud Kubernetes Service. Although the toolchain uses the reviews application from Bookinfo by default, you can link to your own GitHub repo instead. This toolchain is preconfigured for continuous delivery, Vulnerability Advisor, source control, issue tracking, and online editing.  Try the tutorial: ["Progressively roll out your application in Kubernetes by using the iter8 toolchain"](https://www.ibm.com/cloud/garage/tutorials/canary-test-kubernetes-iter8-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud Kubernetes Service (Kubernetes cluster) |
| ["Develop a Kubernetes app with image signing" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fimage-signed-secure-kube-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can securely develop and deploy an application into a Kubernetes cluster that is managed by the IBM Cloud Kubernetes Service. This toolchain requires pre-configured keys that you can set up by using the [Key-Management-Admin-toolchain template](https://github.com/open-toolchain/key-management-admin-toolchain). Although the toolchain uses a sample Node.js Hello World app by default, you can link to your own GitHub repo instead. This toolchain is pre-configured with Docker image signing and a signature check policy on the Kubernetes cluster for continuous delivery with Vulnerability Advisor, source control, issue tracking, and online editing.  Try the tutorial: ["Develop a Kubernetes app with Secure Image Signing" toolchain](https://www.ibm.com/cloud/architecture/tutorials/develop-a-kubernetes-app-with-secure-image-signing) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud Kubernetes Service (Kubernetes cluster) |
| ["Develop and test a Cloud Foundry app" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fdra-toolchain-demo)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this cloud-native toolchain, you can use DevOps Insights to gate the deployment of a simple Cloud Foundry application. By default, the toolchain uses a sample Node.js weather app, or you can link to your own GitHub repository. The toolchain runs unit tests using Mocha and checks code coverage by using Istanbul.  Try the tutorial: [Use the "Develop and test a Cloud Foundry app" toolchain](https://www.ibm.com/cloud/garage/tutorials/use-develop-test-cloud-foundry-app-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  Git Repos and Issue Tracking  IBM Cloud® DevOps Insights |
| ["Develop and test microservices on Cloud Foundry" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fmicroservices-toolchain-hosted)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this cloud-native toolchain, you can use a sample to create an online store that consists of three microservices: a Catalog API, an Orders API, and a UI that calls both of the APIs. The toolchain is preconfigured for continuous delivery, source control, functional testing, issue tracking, online editing, and alert notification.  Try the tutorial: [Use the "Develop and test microservices on Cloud Foundry" toolchain](https://www.ibm.com/cloud/garage/tutorials/use-develop-test-microservices-on-cloud-foundry-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  GitHub and Issues  IBM Cloud  IBM Cloud® DevOps Insights  PagerDuty  Sauce Labs  Slack |
| ["Garage Method tutorial with Cloud Foundry" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fcloud-native-toolchain-tutorial).  Available in Dallas, Washington, Toronto, Sao Paulo, Tokyo, Sydney, Osaka, Frankfurt, and London | This toolchain demonstrates the DevOps practices that are featured in the Garage Method. The toolchain is preconfigured for continuous delivery, source control, test automation, and automated monitoring and operations. It comes with a sample app that is written in Node.js Express 4, which you can further extend.  Try the tutorial: [Become a Garage Method advocate](https://www.ibm.com/cloud/garage/content/course/gm_advocate) | Delivery Pipeline  Eclipse Orion Web IDE  GitHub and Issues  Google Analytics  IBM Cloud  New Relic  PagerDuty  Sauce Labs  Slack |
| ["DevOps Insights Quick Start Demo" toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fdevops-insights%2FDevOpsInsights_Demo_Toolchain_Template)  Available in Dallas, Washington, Toronto, Sao Paulo, Frankfurt, Tokyo, Sydney, Osaka, and London | With this toolchain, you can explore DevOps Insights, with no setup required. To get started, log in to IBM Cloud. This demonstration contains data from a reference toolchain and three GitHub repos. Explore how to organize, test, build, and deploy data for all applications, from all teams, within the Quality Dashboard. Evaluate trends and understand areas that need improvements so that you know where to focus your resources.  Try the tutorial: [Explore IBM Cloud® DevOps Insights](https://www.ibm.com/cloud/garage/tutorials/explore-ibm-cloud-devops-insights) | GitHub and Issues  IBM Cloud® DevOps Insights |
| [Build your own toolchain](https://cloud.ibm.com/devops/setup/deploy?repository=https%3A%2F%2Fgithub.com%2Fopen-toolchain%2Fempty-toolchain)  Available in Dallas, Washington, Toronto, Sao Paulo, Tokyo, Sydney, Osaka, Frankfurt, and London | This toolchain has no preconfigured tools. If you are already familiar with toolchains, you can set up your own toolchain.  Try the tutorial: [Create a custom toolchain](https://www.ibm.com/cloud/garage/tutorials/create-a-custom-toolchain) | None |
| Continuous Delivery toolchain  Available in Dallas, Washington, Tokyo, Sydney, Frankfurt, and London | This toolchain is used when you enable continuous delivery for an app.  Try the tutorials:  [Add a toolchain to an app](https://www.ibm.com/cloud/garage/tutorials/add-a-toolchain-to-an-app)  [Create a custom toolchain](https://www.ibm.com/cloud/garage/tutorials/create-a-custom-toolchain) | Delivery Pipeline  Eclipse Orion Web IDE  GitHub and Issues  IBM Cloud |
| Custom toolchain template  Available in Dallas, Washington, Tokyo, Sydney, Frankfurt, and London | You can create a custom toolchain template that can be used by others.  See the documentation: [Creating custom toolchain templates](https://github.com/open-toolchain/sdk/wiki)  Try the tutorial: [Create a template for a custom toolchain](https://www.ibm.com/cloud/garage/tutorials/create-a-template-for-a-custom-toolchain) |  |

## Develop a Cloud Foundry app toolchain



Reference: <https://www.ibm.com/cloud/architecture/images/toolchain_icons/simple_v2_tc.svg>

# Creating toolchains

A toolchain is a set of tool integrations that support development, deployment, and operations tasks. The collective power of a toolchain is greater than the sum of its individual tool integrations.

Open toolchains are available on IBM Cloud®. You can create a toolchain in two ways: use a template to create a toolchain or create a toolchain from an app.

Each toolchain is associated with a specific resource group or organization (org). If a toolchain is associated with a resource group, any user that has Identity and Access Management (IAM) Viewer permission for the toolchain resource or the resource group that contains it can access the toolchain. If the toolchain is associated with an org, any user that is a member of that org can be added to the access control list for any of its associated toolchains. For more information about access control for toolchains in Cloud Foundry orgs, see [Managing user access to toolchains in Cloud Foundry orgs](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains-cf-security). For more information about access control for toolchains in resource groups, see [Managing user access to toolchains in resource groups](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains-iam-security).

On IBM Cloud Public, depending on the template or toolchain that you use, the toolchain might include a GitHub or Git repository (repo) that is populated with app starter code and a preconfigured delivery pipeline. When you push changes to the toolchain's repo, the delivery pipeline automatically builds and deploys the app to IBM Cloud.

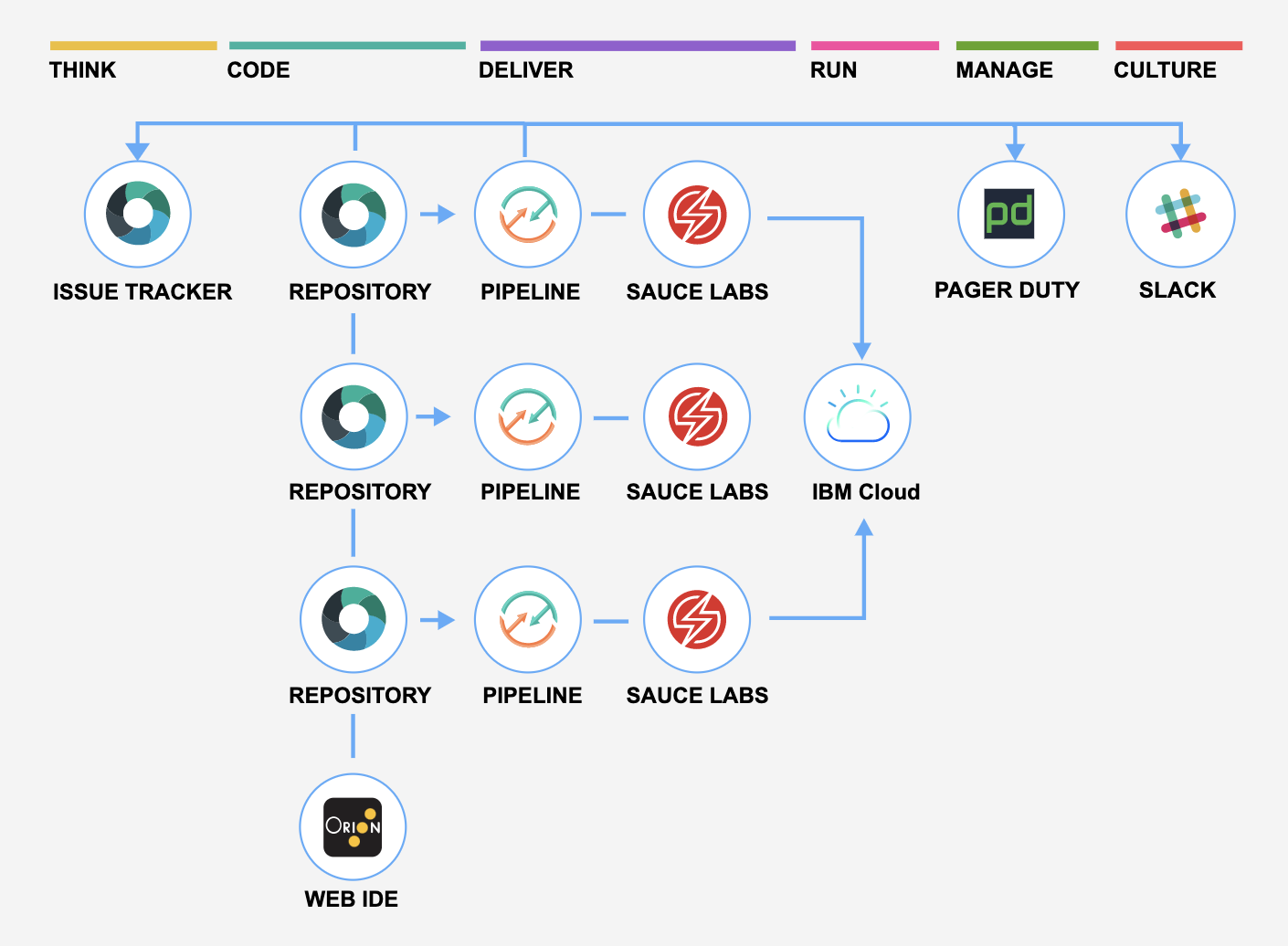
To see which toolchains and tool integrations are available, see [Toolchain availability, templates, and tutorials](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-cd_about).

## Creating a toolchain from a template

You can use a template as a starting point to [create a toolchain](https://cloud.ibm.com/devops/create) that includes a specific set of tool integrations. Depending on the template that you use, you can create a toolchain that has a specific set of tool integrations or an empty toolchain that you can add tool integrations to. Learn more about how to use the templates from the [IBM Cloud Garage Method](https://www.ibm.com/cloud/garage/category/tools).

1. Log in to [IBM Cloud](http://cloud.ibm.com/).
2. From the IBM Cloud console, click the menu icon , and select **DevOps**.
3. On the **Toolchains** page, click **Create a Toolchain**.
4. On the **Create a Toolchain** page, click a toolchain template.
5. Review the diagram of the toolchain that you are about to create. The diagram shows each tool integration in its lifecycle phase in the toolchain.

The diagram in the following image is an example. When you create a toolchain, the diagram shows each tool integration that is part of the toolchain.

Figure 1. Sample Toolchain

https://cloud.ibm.com/docs-content/v1/content/540117727afb925078cc271111e51342cdea6469/ContinuousDelivery/images/toolchain\_diagram2.png

1. Review the default information for the toolchain settings:
   * The toolchain's name identifies it in IBM Cloud. If you want to use a different name, change the toolchain's name.
   * The region to create the toolchain in. If you want to use a different region, select it from the list of available regions.
   * The resource group to create the toolchain in. If you want to use a different resource group, select it from the list of available resource groups.
   * The provider for your source repository, such as GitHub, GitLab, or Bitbucket. If you want to use a different source provider, select it from the list of available repos.
2. In the Tool Integrations section, select each tool integration that you want to configure for your toolchain. A few of the tool integrations do not require configuration. For more information about configuring the tool integrations, see [Configuring tool integrations](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-integrations).

A secret is credentials that can be provided in the form of a password, authentication token, API key, or certificate. For example, when you add the Delivery Pipeline tool integration to your toolchain, you must provide a secret in the form of an API key.

a. In the Tool Integrations section, click **Delivery Pipeline**.

b. Click **New** to create an IBM Cloud® API Key.

c. Click **OK** to apply your new API key.

Each user can have a maximum of 20 API keys.

d. Click **OK** to create the API key without saving a secure copy of the key.

e. To securely save the API key so that you can use it again in other toolchains and starter kit workflows:

* + Select the **Save this key in a secrets store for reuse** checkbox to integrate with the default IBM Key Protect secrets store.
  + If you don't have an existing instance of Key Protect, specify a name for the instance and the secret.
  + Click **OK** to apply your new API key.

f. To validate that your Key Protect instance was successfully created, go to your [IBM Cloud® Resource list](https://cloud.ibm.com/resources) and expand the **Services** twistie. To view your API keys, from the menu bar, click **Manage** > **Access (IAM)**, and select **IBM Cloud® API keys**.

g. The API key that you created and copied to Key Project is now available for use on the Create a Toolchain page. Any tool integration that requires a secret displays a key icon. Click the key icon to open a Secrets Picker dialog box to retrieve secrets from one or more Key Protect instances.

1. Click **Create**. Several steps run automatically to set up your toolchain. The tool integrations that are set up are different depending on which toolchain template you selected. For example, when you create a Microservices toolchain on IBM Cloud Public, these steps are run:
   * The toolchain is created.
   * If you configured Delivery Pipeline, the pipelines are created and triggered.
   * If you configured Sauce Labs, the toolchain is set up to add Sauce Labs test jobs to the pipelines.
   * If you configured PagerDuty, the toolchain is set up to send alert notifications to the PagerDuty service that you specified.
   * If you configured Slack, the toolchain is set up to send notifications about deployment status to the Slack channel that you specified.
   * If you configured a source code tool integration such as GitHub, the sample GitHub repo is cloned into your GitHub account.

## Creating a toolchain from an app

You can create a toolchain from your app. The toolchain can support continuous development, deployment, monitoring, and more, and it is associated with your app. Each app can be associated with a toolchain. When you push changes to the toolchain's GitHub repo, the pipeline automatically builds and deploys the app.

1. If you created your app by using a starter kit, click **Deploy my app** on your app's details page. Next, select a deployment target. If you use IBM Cloud Public, your app is configured for continuous delivery from a new GitHub repo that is populated with the app starter code.
2. On the toolchain configuration page, review the diagram of the toolchain that you are about to create. The diagram shows each tool integration in its lifecycle phase in the toolchain.
3. Review the default information for the toolchain settings. The toolchain's name identifies it in IBM Cloud. If you want to use a different name, change the toolchain's name.
4. In the Tool Integrations section, select each tool integration that you want to configure for your toolchain. A few of the tool integrations do not require configuration. For more information about configuring the tool integrations, see [Configuring tool integrations](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-integrations).
5. Click **Create**. Several steps run automatically to set up your toolchain. For example, when you create a toolchain from an app on IBM Cloud Public, these steps are run:
   * The toolchain is created.
   * If you configured Delivery Pipeline, the pipelines are created and triggered.
   * If you configured GitHub, the sample GitHub repo is cloned into your GitHub account.

## Viewing a toolchain

### **Viewing a toolchain in the console**

After you configure the toolchain and its tool integrations, you can view a visual representation of the toolchain.

1. From the IBM Cloud console, click the menu icon , and select **DevOps**.
2. On the Toolchains page, select a **Resource Group** or **Location**. All of the toolchains that are contained within the selected resource group or Cloud Foundry org are displayed. Click the toolchain that you want to view to open its Overview page. Alternatively, on the App details page in your app, click the toolchain name.
3. To access a tool integration that is in your toolchain, go to the appropriate card, such as **Delivery pipelines**, and then click the tool integration.

**IBM Cloud DevOps Services**

### **Continuous Delivery**

# Getting started with Continuous Delivery

Adopt a DevOps or DevSecOps approach by using IBM Cloud® Continuous Delivery, which includes open toolchains that automate the building and deployment of applications. You can get started by creating a simple deployment toolchain that supports development, deployment, and operations tasks.

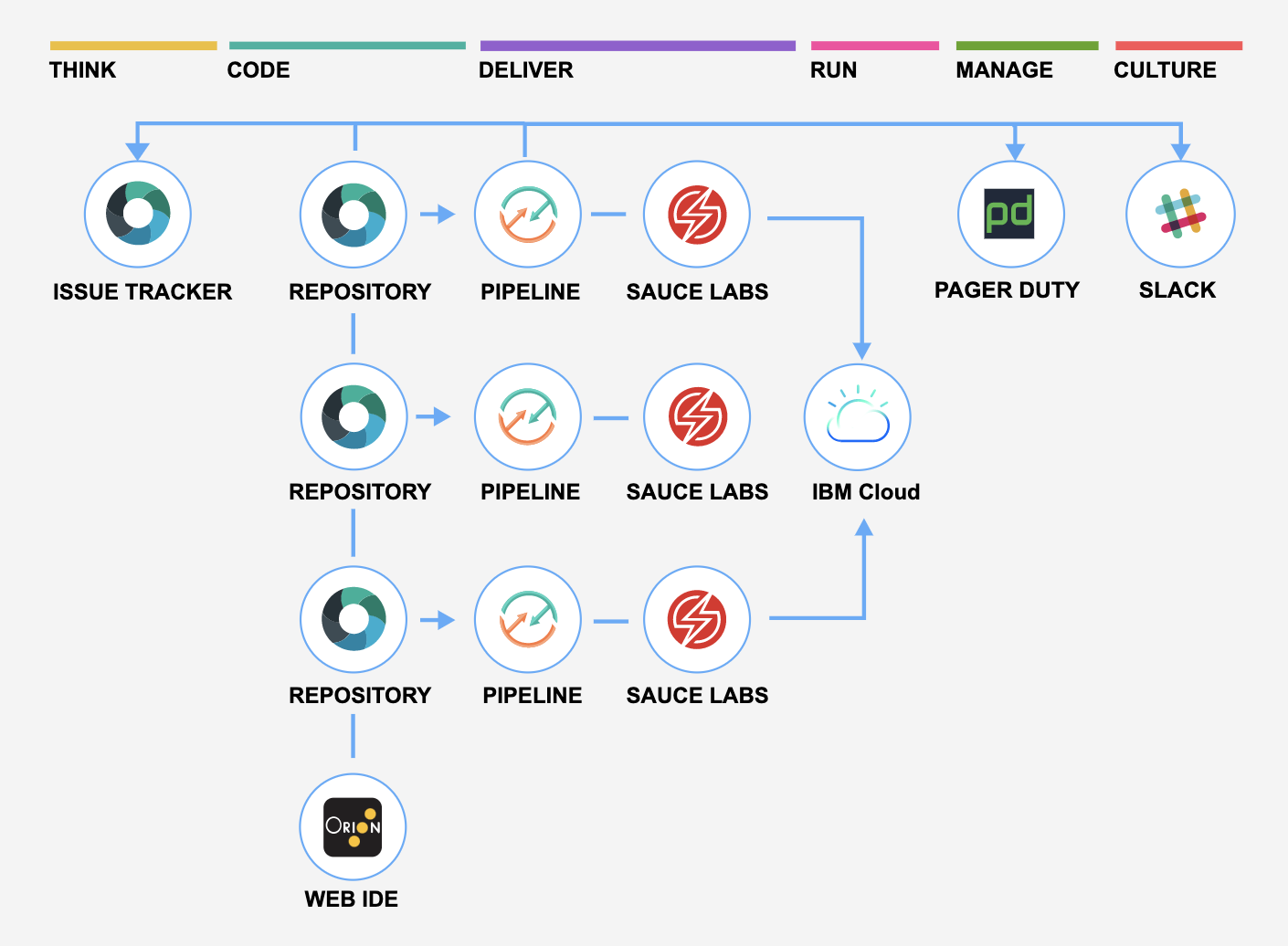
## Prerequisites

Before you can create a continuous delivery toolchain from a template, you must create an instance of Continuous Delivery by selecting it from the IBM Cloud catalog. The toolchain integrates tools for planning, developing, deploying pipelines, and managing your applications. You can always add or remove tools from your toolchains. If you already have toolchains, you can [view existing toolchains](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains_getting_started#viewing_a_toolchain). For more information about working with toolchains, see [Using toolchains](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains-using).

## Step 1: Select a toolchain template

1. On the **Create a Toolchain** page, click a [toolchain template](https://cloud.ibm.com/devops/create).
2. Review the diagram of the toolchain that you are about to create. The diagram shows each tool integration in its lifecycle phase in the toolchain.

The diagram in the following image is an example. When you create a toolchain, the diagram shows each tool integration that is part of the toolchain.

Figure 1. Toolchain diagram

## Step 2: Create a toolchain

1. Review the default information for the toolchain settings:
   * The toolchain's name identifies it in IBM Cloud. If you want to use a different name, change the toolchain's name.
   * The region to create the toolchain in. If you want to use a different region, select it from the list of available regions.
   * The resource group to create the toolchain in. If you want to use a different resource group, select it from the list of available resource groups.
   * The provider for your source repository, such as GitHub, GitLab, or Bitbucket. If you want to use a different source provider, select it from the list of available repos.
2. In the Tool Integrations section, select each tool integration that you want to configure for your toolchain. A few of the tool integrations do not require configuration. For information about configuring the tool integrations, see [Configuring tool integrations](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-integrations).
3. Click **Create**. Several steps run automatically to set up your toolchain. The tool integrations that are set up are different depending on which toolchain template you selected. For example, when you create a Microservices toolchain on IBM Cloud Public, these steps are run:
   * The toolchain is created.
   * If you configured Delivery Pipeline, the pipelines are created and run.
   * If you configured Sauce Labs, the toolchain is set up to add Sauce Labs test jobs to the pipelines.
   * If you configured PagerDuty, the toolchain is set up to send alert notifications to the PagerDuty service that you specified.
   * If you configured Slack, the toolchain is set up to send notifications about deployment status to the Slack channel that you specified.
   * If you configured a source code tool integration such as GitHub, the sample GitHub repo is cloned into your GitHub account.

### **Delivery Pipeline**

# Classic Delivery Pipeline overview

IBM Cloud® Continuous Delivery includes the Classic Delivery Pipeline to build, test, and deploy in a repeatable way with minimal human intervention. In a pipeline, sequences of stages retrieve input and run jobs, such as builds, tests, and deployments.

Your permissions to view, modify, or run a pipeline are based on the access control for the toolchain that owns the pipeline. For more information about access control for toolchains, see [Managing access to toolchains in resource groups](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-toolchains-iam-security) and [Managing access to toolchains in Cloud Foundry orgs](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-toolchains-cf-security).

You can specify the scripts to run in many of the job types that are provided by the pipeline, giving you direct control over what is run by the job. These scripts run in a Docker image that contains a number of standard development tools, including tools that are required for interacting with the IBM Cloud runtimes. For more information about what the standard Docker image contains, see [Preinstalled resources](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-deliverypipeline_environment#deliverypipeline_resources). If your job requires development tools that are not available in the standard image, or you need different versions of those tools, you can use a custom image. For more information about custom images, see [Working with custom Docker images](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-custom_docker_images#custom_docker_images).

When the pipeline runs scripts, properties that describe the context where the job is running are passed to the script by using environment variables. For example, the URL of the repo that is the input to the stage, the name of the stage and the job that is being run, the parameters specified by the job type, and so on. To view a list of the available environment variables, see [Preinstalled resources](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-deliverypipeline_environment#deliverypipeline_resources).

You can define properties at both the pipeline level and the stage level. Pipeline properties are shared across all stages and jobs in a pipeline. Stage properties are unique to a particular stage, and shared across all jobs in that stage. For more information about properties, see [Environment properties (Environment variables)](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-deliverypipeline_about#environment_properties).

## Stages

Stages organize input and jobs as your code is built, deployed, and tested. Stages accept input from either source control repositories (SCM repositories) or build jobs in other stages. For SCM repositories, the input is the contents of a particular branch in the repository; for build jobs, the input is the artifacts that are produced by the job. When you create your first stage, the **INPUT** tab contains default settings.

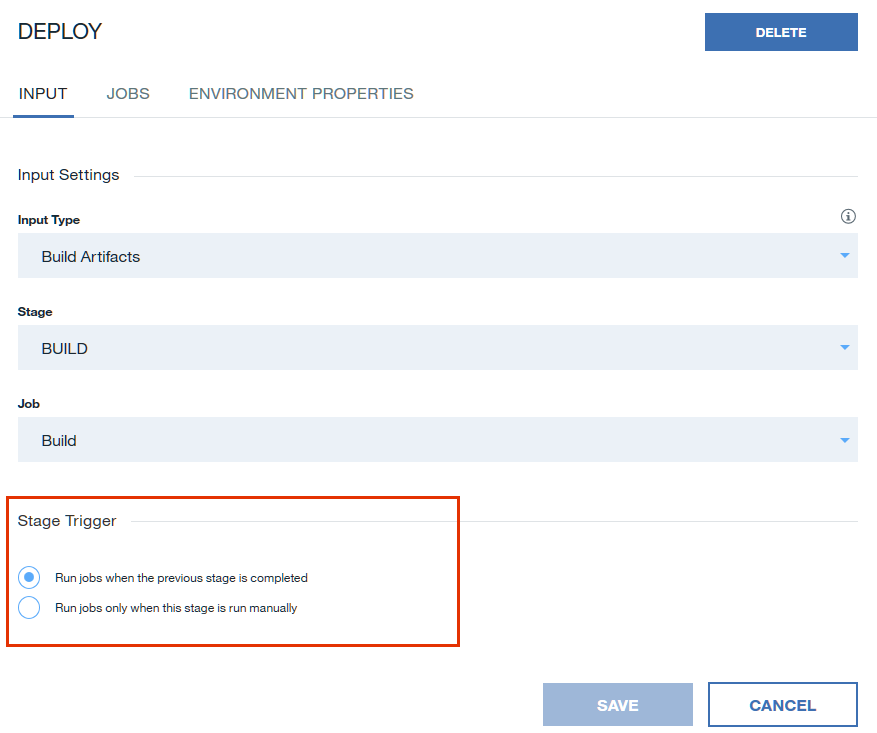
Similar to how you can define pipeline properties, you can also define stage properties for use in all of the jobs in a particular stage. For example, you might define a TEST\_URL property that passes a URL to the deploy and test jobs in a stage. The deploy job deploys to that URL and the test job tests the running app at the URL. Stage properties are also passed to job scripts by using environment variables. If the same property is defined at both the pipeline level and the stage level, the value of the stage property is used.

By default in a stage, builds and deployments are run automatically every time that changes are delivered to a project's SCM repository. Stages and jobs run serially; they enable flow control for your work. For example, you might place a test stage before a deployment stage. If the tests in the test stage fail, the deployment stage does not run.

The Delivery Pipeline uses public and private workers to run the jobs in a stage. By default, pipeline jobs are run by using public workers on IBM-managed public shared infrastructure.

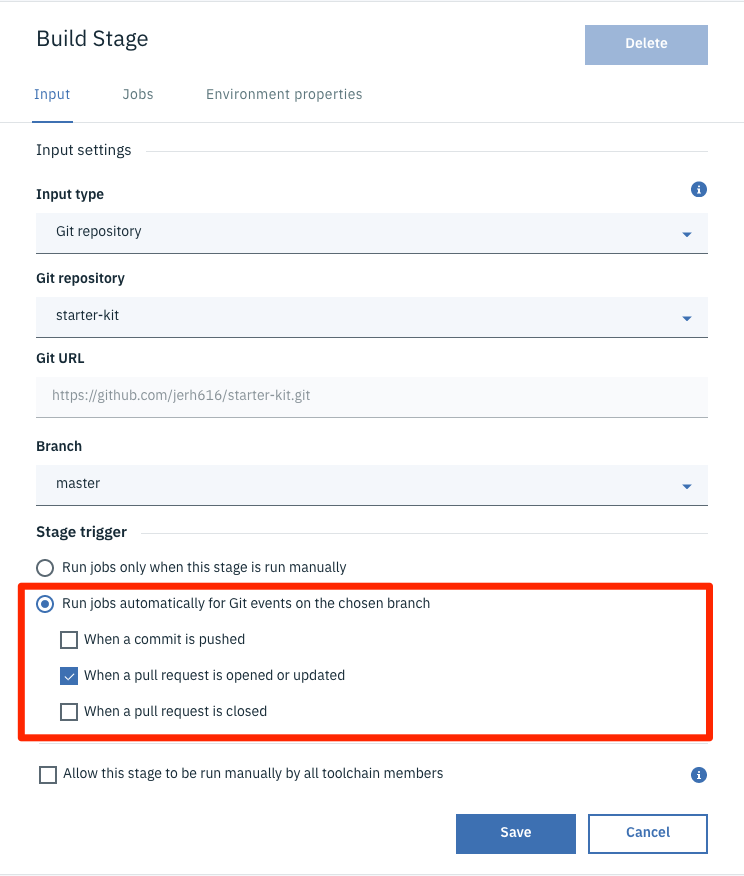
In certain scenarios, your Delivery Pipeline might require access to internal or on-premises resources. In these situations, you can connect to and integrate a Delivery Pipeline Private Worker to run on your own Kubernetes infrastructure.

You might want tighter control of a specific stage. If you do not want a stage to run every time that a change occurs at its input, you can disable the capability. On the **INPUT** tab, in the Stage Trigger section, click **Run jobs only when this stage is run manually**.

Figure 1. Input tab

More stage trigger options are available for stages that use the Git repository input type. For example, you can choose to run jobs automatically for Git events on the chosen branch. When you choose this trigger type, you must select one or more of the following event types:

* **When a commit is pushed** triggers when a push is made to the selected repo branch.
* **When a pull/merge request is opened or updated** triggers when a pull request or merge request is opened or edited.
* **When a pull/merge request is closed** triggers when a pull request or merge request is closed, even without an associated commit.

Figure 2. Input tab triggers

If you select the **When a pull/merge request is opened or updated** checkbox, the status of the pipeline is returned to the Git repo. When a pull request or merge request triggers your pipeline, an inline status check is displayed on the page. A status check is displayed for each of the stages that are run in your pipeline, and links to the logs and history for each stage are provided. As the status check runs, it updates from pending to either successful or failed. If your pipeline contains multiple stages, each stage reports its status in the check list.

You can also restrict merging based on the results of the status checks by using Git branch protection rules. After a branch protection rule is created, all merging is blocked until all of the required status checks are successful.

### **Bitbucket Cloud pull requests**

Bitbucket Cloud currently does not support repository references for pull requests, which is required by the Continuous Delivery service. This feature allows pull requests to be sent to the repo that you want to access by using references in the following format: refs/pull/123/…

You can [locally fetch and check out a pull request](https://confluence.atlassian.com/bbkb/how-to-locally-fetch-and-checkout-a-pull-request-724402529.html) by using the source repo URL. However, if the source repo is a private forked repo, the Continuous Delivery service does not have the access that is required to manage pull requests. To work around this limitation, you must explicitly provide the required access to the forked repo in the pipeline script.

In the following sample bash pipeline script, two users are using Bitbucket Cloud and they each have a private fork of their main repo (bitbucket.org/userA/repo-forked-A and bitbucket.org/userB/repo-forked-B). The script is set up to check out the pull request when a build job is triggered by a pull request open event or update event from one of the two forked repos.

case "$BITBUCKET\_PR\_SOURCE\_HOST" in #BITBUCKET\_PR\_SOURCE\_HOST is an environment exported by pipeline if job is triggered by a bitbucket pull request

\*userA\*) #userA should be replaced to anything to identify a forked repo's url

url="https://$username:$password@$BITBUCKET\_PR\_SOURCE\_HOST" #you need to provide username and password for repo-forked-A

;;

\*userB/repo-forked-B\*) #userB/repo-forked-B should be replaced to anything to identify a forked repo's url

url="https://$username1:password1@$BITBUCKET\_PR\_SOURCE\_HOST" #you need to provide username1 and password1 for repo-forked-B

;;

esac

git fetch $url $BITBUCKET\_PR\_SOURCE\_BRANCH #BITBUCKET\_PR\_SOURCE\_BRANCH is an environment exported by pipeline if job is triggered by a bitbucket pull request

git checkout FETCH\_HEAD

Show more

### **Build stage**

The build stage specifies a **Builder type** to indicate how to build the artifacts.

The following Builder types are available:

| **Builder type** | **Description** | **Supported job types** |
| --- | --- | --- |
| Simple | Archives the current stage's input without modification for use by future stages. Typically, this builder type is useful only when the stage's input is from an SCM repository. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version. |
| Ant | Uses Apache Ant files to manage the build job. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Container registry | Builds docker images and uploads them to the IBM Cloud Container Registry. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **API key**: The IBM Cloud API key to use to provide permissions to account resources.  **Container Registry namespace**: The namespace where you want to store your built image.  **Docker image name**: The name of the image that this job builds and uploads to the IBM Cloud Container Registry.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Custom Docker image | Builds by using your custom docker image with fine-grained control over the versions of node, Java™, or other tools. | **Docker image name**: The name of the image that this job builds and uploads to the IBM Cloud Container Registry.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Build archive directory**: Specifies the directory that contains the job's output to be archived for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Gradle | Builds by using Gradle. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in to your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory** - Specifies the directory that contains the job's output to be archived for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Grunt | Builds by using Grunt. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in to your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Maven | Builds by using Apache Maven. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in to your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| npm | Installs dependencies with the Node package manager. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Shell script | Runs a UNIX shell script, such as Bash. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Build script**: Runs in a new Ubuntu shell whenever the job runs. In the script field, enter a script or reference scripts that are stored in to your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage.  **Enable test report**: Select this checkbox to specify that the build job runs tests that produce result files in JUnit XML format. A report based on the result files is displayed on the Tests tab of the Job Results page. If any tests fail, the job is marked as failed.  **Enable code coverage report**: Select this checkbox to show more fields that you can use for the code coverage report. You can specify the Coverage runner (such as Istanbul, JaCoCo, and Cobertura), the location of the Coverage result file, and the Coverage result directory, relative to the Working directory. |
| Gradle (Artifactory, Nexus, or SonarQube) | Builds and deploys by using Gradle with a Nexus or Artifactory repository. Gradle also integrates with SonarQube. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Repository tool integration instance**: The name of the repository tool integration instance to use with this build job.  **Repository tool integration type**: The type of tool integration to get Gradle information from.  **SonarQube integration instance**: The name of the SonarQube integration instance to use with this build job.  **Build command**: The build command to run whenever the job runs. In the **Script** field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage. |
| Maven (Artifactory, Nexus, or SonarQube) | Builds and deploys by using Maven with a Nexus or Artifactory repository. Maven also integrates with SonarQube. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Repository tool integration instance**: Name of the repository tool integration instance to use with this build job.  **Repository tool integration type**: Type of tool integration to get Gradle information from.  **SonarQube integration instance**: Name of the SonarQube integration instance to use with this build job.  **Build command**: Build command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage. |
| npm (Artifactory or Nexus) | Builds by using npm with a Nexus or Artifactory repository. | **Pipeline image version**: Runs in a container by using a built-in docker image, which provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Repository tool integration instance**: The name of the repository tool integration instance to use with this build job.  **Repository tool integration type**: The type of tool integration to get Gradle information from.  **SonarQube integration instance**: The name of the SonarQube integration instance to use with this build job.  **Build command**: The build command to run whenever the job runs. In the **Script** field, enter a script or reference scripts that are stored in your project’s source control.  **Increment snapshot module version**: Supports continuous delivery by incrementing the module version locally based on the contents of the package.json file and the current reported snapshot in the npm registry at the publish step.  **Working directory**: Specifies the directory where the script is run.  **Build archive directory**: Specifies the directory that contains the job's output to archive for use by a subsequent stage. |

### **Deploy stage**

The deploy stage specifies input from a Build stage. The jobs in the deploy stage specify a **Deployer type**. The following Deployer types are available:

| **Deployer type** | **Description** | **Supported job types** |
| --- | --- | --- |
| Cloud Foundry | Deploys applications to Cloud Foundry servers, such as IBM Cloud. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Cloud Foundry Type**: The type of environment you want to deploy to.  **API key**: The IBM Cloud API key to use to provide permissions to account resources.  **Application name**: The name that is assigned to the application during deployment. This name is assigned to the environment variable and is referenced in the IBM Cloud script.  **Deploy script**: Deploy command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control. |
| Custom Docker image | Deploys by using your custom Docker image with fine-grained control over the versions of node, Java™, or other tools. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Cloud Foundry Type**: The type of environment you want to deploy to.  **API key**: The IBM Cloud API key to use to provide permissions to account resources.  **Docker image name**: The name of the image that this job builds and uploads to the IBM Cloud Container Registry.  **Deploy script**: Deploy command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control. |
| Kubernetes | Deploys applications to Kubernetes clusters, such as those found within the IBM Cloud Container Service. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **API key**: The IBM Cloud API key to use to provide permissions to account resources.  **Cluster name**: Name of the Kubernetes cluster; the platform that you deploy your Kubernetes components on.  **Deploy script**: Deploy command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control. |

### **Test stage**

The test stage specifies the test configuration. The jobs in the test stage specify a **Tester type**. The following Tester types are available:

| **Tester type** | **Description** | **Supported job types** |
| --- | --- | --- |
| Simple | Launches a shell command to run the automated tests, with an optional test report. | **Pipeline image version**: Not used.  **Test script**: Test command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: The directory where the test script is run.  **Enable test report**: Not used.  **Enable code coverage report**: Not used. |
| Custom Docker image | Tests by using your custom Docker image with fine-grained control over the versions of node, Java™, or other tools. | **Docker image name**: The name of the Docker image to run the job with. To make sure that your jobs run in a clean context, run them in Docker containers.  **Test script**: Test command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: The directory where the test script is run.  **Enable test report**: Not used.  **Enable code coverage report**: Not used. |
| Vulnerability Advisor | Runs a compliance and vulnerability check against the specified image, and displays the results. If any issues are found, this stage fails. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **API key**: The IBM Cloud API key to use to provide permissions to account resources.  **Container Registry namespace**: The namespace where your built image is stored.  **Docker image name**: The name of the Docker image to run the job with. To make sure that your jobs run in a clean context, run them in Docker containers.  **Docker image tag**: A tag for the Docker image that is displayed in the IBM Cloud Container Registry.  **Test script**: Test command to run whenever the job runs. In the script field, enter a script or reference scripts that are stored in your project’s source control.  **Working directory**: The directory where the test script is run.  **Enable test report**: Not used.  **Enable code coverage report**: Not used. |
| Sauce Labs | Runs JavaScript, Node, or Java™ tests by using Sauce Labs. | **Pipeline image version**: Runs in a container by using a built-in docker image that provides various built-in commands. To adopt newer versions of those commands, use a newer image version.  **Service instance**: Select a configuration instance or create one. |

### **Deprecated job types**

Several job types, such as the IBM Globalization Pipeline Build job, the Simplified Cloud Foundry Org Build job, the Space Shell Test job, and the DevOps Insights Gate Test job are deprecated. Although these job types are deprecated, you might still be able to load them in the UI, with an indicator that the job type is deprecated. Alternatively, your job might revert to another job type that is still supported, with a warning notification.

If you need to use the configuration from a deprecated job type, use one of the following methods to access the pipeline configuration.

* Use the IBM Cloud Devtool:

ic dev pipeline-get 7325f511-492a-4c35-a388-5e499e65d6bb -output JSON

* Use the Delivery Pipeline API:

curl --location --request GET 'https://devops-api.us-south.devops.cloud.ibm.com/v1/pipeline/pipelines/7325f511-492a-4c35-a388-5e499e65d6bb/stages' --header 'Authorization: Bearer <IAM Bearer token>

* From the **Network** tab of the Delivery Pipeline UI, filter by the pipeline ID to locate the pipeline that contains the deprecated job type data.

### **API keys**

Some of the standard pipeline jobs use IBM Cloud API keys to access services, such as deploying to Cloud Foundry and Kubernetes. The [IBM Cloud Identity and Access Management (IAM)](https://cloud.ibm.com/docs/services/account?topic=account-iamoverview) service provides two types of API keys:

* **user API keys**: These API keys provide full access to all of the services and resources that the user has access to.
* **service API keys**: You can configure service API keys to provide specific access to various services and resources.

Because pipeline jobs run user-created scripts that might use service API keys in arbitrary ways, the pipeline cannot determine the set of restrictions to apply to a particular key. In such cases, if you request that the pipeline creates an API key, it creates a user API key. To maintain strong security, instead use a service API key with access that is restricted to only the services and resources that you need in the script. In this instance, you must create the API key yourself. For more information about creating an API key, see [IBM Cloud API keys](https://cloud.ibm.com/docs/account?topic=account-userapikey#create_user_key).

## Jobs

A job is an execution unit within a stage. A stage can contain multiple jobs, and the jobs in a stage run sequentially. By default, if a job fails, subsequent jobs in the stage do not run.

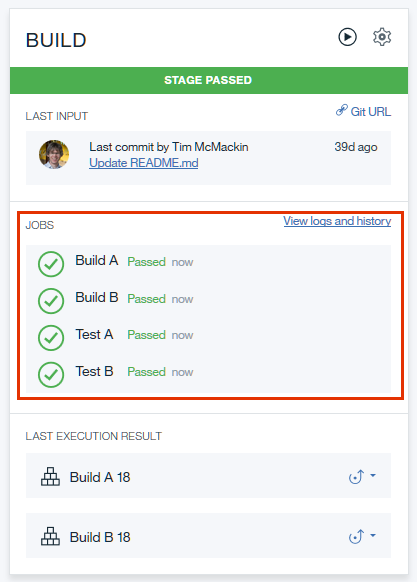


Figure 3. Build and test jobs within a stage

Jobs run in discrete working directories within Docker containers that are created for each pipeline run. Before a job is run, its working directory is populated with input that is defined at the stage level. For example, you might have a stage that contains a test job and a deploy job. If you install dependencies on one job, they are not available to the other job. However, if you make the dependencies available in the stage's input, they are available to both jobs.

Except for Simple-type build jobs, when you configure a job, you can include UNIX shell scripts that include build, test, or deployment commands. Because jobs are run in ad hoc containers, the actions of one job cannot affect the run environments of other jobs, even if those jobs are part of the same stage.

Sample build and deploy scripts can be found in <https://github.com/open-toolchain/commons>.

Additionally, pipeline jobs can run only the following commands as sudo:

* /usr/sbin/service
* /usr/bin/apt-get
* /usr/bin/apt-key
* /usr/bin/dpkg
* /usr/bin/add-apt-repository
* /opt/IBM/node-v0.10.40-linux-x64/npm
* /opt/IBM/node-v0.12.7-linux-x64/npm
* /opt/IBM/node-v4.2.2-linux-x64/npm
* /usr/bin/Xvfb
* /usr/bin/pip

After a job runs, the container that was created for it is discarded. The results of a job run can persist, but the environment in which it ran does not.

To learn how to add a job to a stage, see [Adding a job to a stage](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-deliverypipeline_build_deploy#deliverypipeline_add_job).

### **Build jobs**

Build jobs compile your project in preparation for deployment. They generate artifacts that can be sent to a build archive directory, although by default, the artifacts are placed in the project's root directory.

Jobs that take input from build jobs must reference build artifacts in the same structure that they were created in. For example, if a build job archives build artifacts to an output directory, a deploy script would refer to the output directory rather than the project root directory to deploy the compiled project. You can specify the directory to archive by entering the directory name in the **Build Archive Directory** field. Leaving the field blank archives the root directory.

When you deploy by using Cloud Foundry, Cloud Foundry includes the correct artifacts to allow your app to run. For more information, see [Deploying applications by using the cf command](https://cloud.ibm.com/docs/cloud-foundry?topic=cloud-foundry-deploy_apps#deploy_apps). The pipeline for a Cloud Foundry app contains a Deploy stage that runs a cf command.

Cloud Foundry tries to [detect the buildpack to use](http://docs.cloudfoundry.org/buildpacks/detection.html). You can specify the [buildpack](https://cloud.ibm.com/docs/cloud-foundry?topic=cloud-foundry-available_buildpacks) to use in the manifest file in the root folder of your app. Buildpacks typically examine user-provided artifacts to determine what dependencies to download and how to configure applications to communicate with bound services. For more information about manifest files, see [Application manifest](https://cloud.ibm.com/docs/cloud-foundry?topic=cloud-foundry-deploy_apps#appmanifest).

### **Deploy jobs**

Deploy jobs upload your project to IBM Cloud as an app and are accessible from a URL. After a project is deployed, you can find the deployed app on your IBM Cloud dashboard.

Deploy jobs can deploy new apps or update existing apps. Even if you first deployed an app by using another method, such as the Cloud Foundry command-line interface or the run bar in the Web IDE, you can update the app by using a deploy job. To update an app, in the deploy job, use that app's name.

You can deploy to one or many regions and services. For example, you can set up your Delivery Pipeline to use one or more services, test in one region, and deploy to production in multiple regions.

### **Test jobs**

If you want to require that conditions are met, include test jobs before or after your build and deploy jobs. You can customize test jobs to be as simple or complex as you need. For example, you might issue a cURL command and expect a particular response. You might also run a suite of unit tests or run functional tests with third-party test services, such as Sauce Labs.

If your tests produce result files in JUnit XML format, a report that is based on the result files is shown on the **Tests** tab of every test result page. If a test fails, the job also fails.

## Environment properties (Environment variables)

A set of predefined environment properties provides access to information about the job's execution environment. For a complete list of the predefined environment properties, see [Environment properties and resources](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-deliverypipeline_environment).

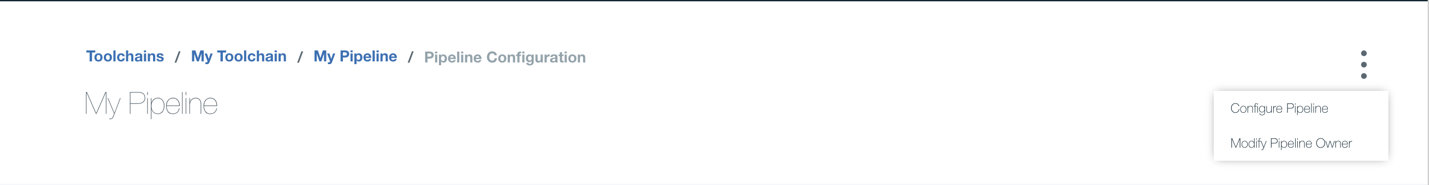
You can also define your own environment properties. For example, you might define an API\_KEY property that passes an API key that is used to access IBM Cloud resources by all scripts in the pipeline.

You can add the following types of properties:

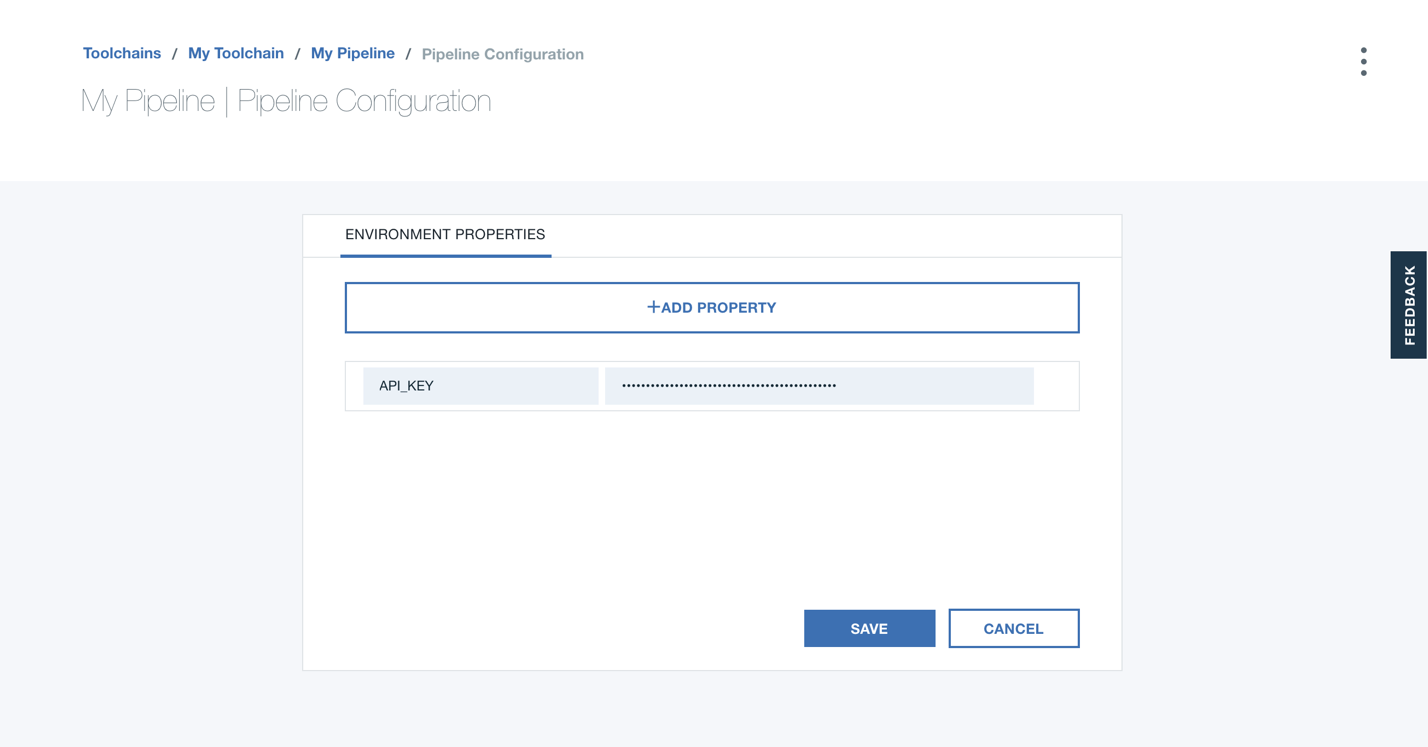
* **Text**: A property key with a single-line value.
* **Text Area**: A property key with a multi-line value. A base64 version of each text area property value is also available. You can access this version by using the property key name with a trailing \_base641 suffix. You can decode the base64 version of a Text Area property and echo it by typing echo "$(echo $multi\_base64 | base64 -d)", where multi is the property key name you defined and multi\_base64 is the additional property that is provided. The pipeline base image contains built in support to manage multi-line encoding transparently. However, if you use a custom image you must append the \_base64 suffix property to prevent issues where your value is truncated by a line-ending.
* **Secure**: A property key with a single-line value that is secured with AES-128 encryption. The value is displayed as asterisks.
* **Properties**: A file in the project's repository. This file can contain multiple properties. Each property must be on its own line. To separate key-value pairs, use the equals sign (=). Enclose all string values in quotation marks. For example, MY\_STRING="SOME STRING VALUE".

### **Pipeline properties**

To define pipeline properties, from the overflow menu on the Pipeline page, select **Configure Pipeline**.

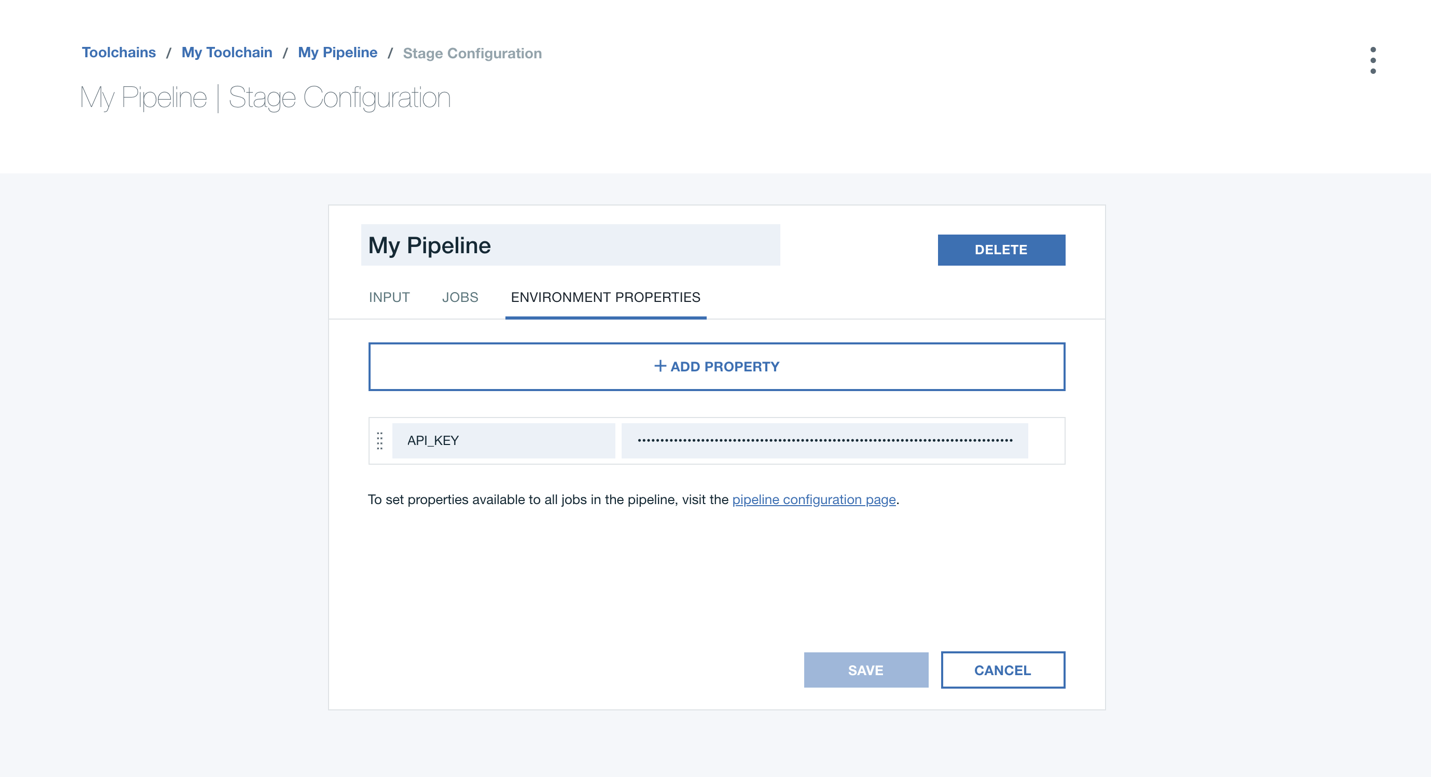
Figure 4. Pipeline overflow menu

From the **ENVIRONMENT PROPERTIES** tab on the Pipeline configuration page, set the pipeline-level environment properties.

Figure 5. Pipeline properties page

### **Stage properties**

To define stage properties, open the Stage configuration page and click the **ENVIRONMENT PROPERTIES** tab.

Figure 6. Stage properties page

### **Computed properties**

You can compute the environment property values that are shared across stages by creating a build.properties file while the stage is running, and then have the next stage run the file. For example, your build job might include the following command in the build script:

echo "IMAGE\_NAME=${FULL\_REPOSITORY\_NAME}" >> $ARCHIVE\_DIR/build.properties

All jobs start by running the build.properties file, if it exists.

## Creating and using artifacts

Build jobs automatically fetch the content in the current folder where the user script is run. If you do not need the entire git repo content for later deployment, it is preferable that you configure an explicit output directory and then copy or create the relevant artifacts there. Job scripts are run in the build result (output directory).

Jobs that deploy to Cloud Foundry need to specify the Platform API key of a user under whose authority jobs run, and the region, org, and space of where to deploy the artifacts. If more services are required to run your app, you must specify them in the manifest.yml file.

Deploy jobs that deploy to the IBM Cloud Kubernetes Service need to specify the Platform API key of a user under whose authority jobs run, a Dockerfile, and optionally a Helm chart.

The job script runs after the job has logged in to the target environment by using the Platform API key that is assigned to it (so that you can run cf push or kubectl commands in the script).

## An example pipeline

A simple pipeline might contain three stages:

1. A Build stage that compiles and runs build processes on an app.
2. A Test stage that deploys an instance of the app and then runs tests on it.
3. A Prod stage that deploys a production instance of the tested app.

This pipeline is shown in the following conceptual diagram:

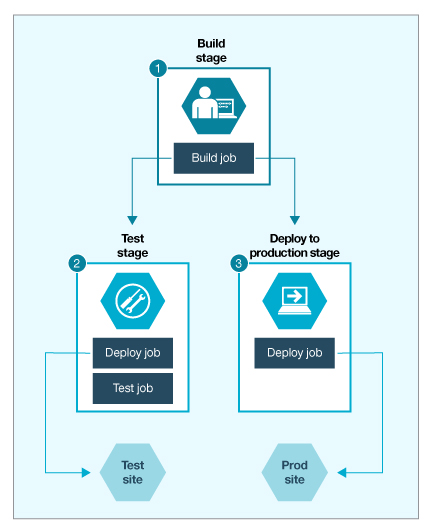


Figure 7. Conceptual model of a three-stage pipeline

Stages take their input from repositories and build jobs, and jobs within a stage run sequentially and independently of each other. In the example pipeline, the stages run sequentially, even though the Test and Prod stages both take the Build stage's output as their input.

## Cloud Foundry Manifest files

Manifest files, which are named manifest.yml and stored in a project's root directory, control how your project is deployed to IBM Cloud. For information about creating manifest files for a project, see the [IBM Cloud documentation about application manifests](https://cloud.ibm.com/docs/cloud-foundry?topic=cloud-foundry-deploy_apps#appmanifest). To integrate with IBM Cloud, your project must have a manifest file in its root directory. However, you are not required to deploy based on the information in the file.

In the pipeline, you can specify everything that a manifest file can do by using cf push command arguments. The cf push command arguments are helpful in projects that have multiple deployment targets. If multiple deploy jobs all try to use the route that is specified in the project manifest file, a conflict occurs.

To avoid conflicts, you can specify a route by using cf push followed by the host name argument, -n, and a route name. By modifying the deployment script for individual stages, you can avoid route conflicts when you deploy to multiple targets.

To use the cf push command arguments, open the configuration settings for a deploy job and modify the **Deploy Script** field. For more information, see the [Cloud Foundry Push documentation](http://docs.cloudfoundry.org/devguide/installcf/whats-new-v6.html#push).

### **Git Repos and Issue Tracking**

Collaborate with your team and manage your source code with a Git repository (repo) and issue tracker that is hosted by IBM and built on [GitLab Community Edition](https://about.gitlab.com/). For more information about GitLab, see the [GitLab documentation](https://us-south.git.cloud.ibm.com/help).

Invite only people that you have a personal or business relationship with to collaborate on a project. Users that use an invitation to a Git repo for purposes other than to collaborate on a project might have their access to the service suspended or revoked.

The [Git Repos and Issue Tracking tool integration](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-grit) supports teams to manage code and collaborate in many ways:

* Manage Git repos through fine-grained access controls that keep code secure
* Review code and enhance collaboration through merge requests
* Track issues and share ideas through the issue tracker
* Document projects on the wiki system

Because this tool integration is built on GitLab Community Edition and hosted by IBM on the IBM Cloud Platform, a few GitLab options are not available. For example, Delivery Pipeline provides continuous integration and continuous delivery for IBM Cloud; therefore, the continuous integration features in GitLab are not supported. In addition, the admin functions are not available because they are managed by IBM.

## Using Git Repos and Issue Tracking with toolchains

You can use a template that contains either a Git Repos and Issue Tracking or GitHub tool integration as a starting point to create a toolchain that you can add Git repositories (repos) to. Alternatively, you can start with an empty toolchain and add either a Git Repos and Issue Tracking or GitHub tool integration to it. By using a toolchain, you can associate Git repos with your resource groups or Cloud Foundry orgs and your Continuous Delivery service instance.

For more information about using toolchains with Git, see [Creating toolchains with Git](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains_git).

## Using Git Repos and Issue Tracking locally

You can locally access the Git repos that are stored in Git Repos and Issue Tracking. For instructions to set up Git locally, see [Start using Git on the command line](https://us-south.git.cloud.ibm.com/help/gitlab-basics/start-using-git).

Git Repos and Issue Tracking supports HTTPS connections only that use TLS1.2. If you use Eclipse to connect, you might be required to specify this protocol for your Java™ version by adding -Dhttps.protocols=TLSv1.2 to your eclipse.ini file and then restarting Eclipse.

## Authenticating with Git Repos and Issue Tracking

Your IBM Cloud login and password are only used to authenticate with Git Repos and Issue Tracking in a web browser. You cannot use your IBM Cloud user credentials to authenticate from external Git clients. To complete remote Git operations, such as clone or push, from your local Git repo, you must use a personal access token or SSH key to authenticate with Git Repos and Issue Tracking.

The display name that appears for you throughout Git Repos and Issue Tracking is populated from your IBM Cloud login information. This name might be visible to other users when they search for users to add to their projects. You can update the name that is displayed for you throughout Git Repos and Issue Tracking from your [Profile page](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-git_working#git_update_name).

### **Creating a personal access token**

To authenticate with your Git repo over HTTPS, you must create a personal access token.

1. On the Git Repos and Issue Tracking User Settings dashboard, on the [Access Tokens page](https://us-south.git.cloud.ibm.com/profile/personal_access_tokens), type the name of the application that you want to create an access token for. For example, Git CLI.
2. Optional: Choose an expiry date for the access token.
3. Select the **api** checkbox to create a personal access token that uses api as the scope.
4. Click **Create Personal Access Token**. Make note of your access token in a secure location for future use.
5. On the [Account page](https://us-south.git.cloud.ibm.com/profile/account), in the Change username section, find your Git Repos and Issue Tracking username. Your username is also displayed as the first segment of the URL for any personal Git repos that you create.
6. Use your Git Repos and Issue Tracking username and personal access token to authenticate with your Git repo from an external Git client.

To learn more, see [Personal access tokens](https://us-south.git.cloud.ibm.com/help/api/README.html#personal-access-tokens).

### **Creating an SSH key**

To create an SSH key, see [Generate an SSH key pair](https://us-south.git.cloud.ibm.com/help/ssh/index.html#generate-an-ssh-key-pair). Accessing your repositories with SSH authentication might require more configuration for proxies and firewalls.

To learn more, see [SSH](https://us-south.git.cloud.ibm.com/help/ssh/index.html).

### **Verifying host key fingerprints**

The first time that you connect to a server by way of Git over SSH, the Git client prompts you to accept the host key fingerprint of the server. You can use the following host key fingerprints to verify SSH connections with the IBM Cloud Git Repos and Issue Tracking servers. Proceed with the connection only if the host key fingerprint matches the specified value for the server that is provided in the following code snippets.

au-syd.git.cloud.ibm.com:

ECDSA:

SHA256:oUpjbxJ+UVIlBvcdcKuprZ0JEtCWkTu1yFTdfFHoEF8

MD5:ca:34:27:f1:49:fd:b4:9d:e8:ce:d2:7b:99:a1:dd:98

ED25519:

SHA256:uUqxTjqUQuBjmQGynGb8pXX6FQ2Ag0VLAh4TtuSZMAQ

MD5:87:ad:c9:26:bd:7f:bc:a8:1c:dc:07:ca:aa:d3:8c:9e

RSA:

SHA256:y+QM+SbgQ7SqzQXqwmJTPD0jni+qsDdqZg/sOgOFWbY

MD5:70:71:95:b5:2a:b4:04:ad:12:b4:77:c6:cf:fe:35:c8

ca-tor.git.cloud.ibm.com:

ECDSA:

SHA256:xqeLs5qKCCNd/SmSTgFktFJW8nTqnF5BmwJSZggguJI

MD5:fb:41:1a:b4:8c:4d:95:c3:67:d9:eb:4a:b1:94:c2:cb

ED25519:

SHA256:mT5EGA/63iaHQZrFkXevP+T/qaFN39JChMGUJtla4nE

MD5:6f:f2:4e:0e:90:0b:2b:e7:fc:f8:d2:1a:16:35:16:fc

RSA:

SHA256:mNvCu12YAUeJVCNfiHNfBKgezh0zgwdwxBs8wXnhPP8

MD5:60:d6:6a:2f:0c:db:52:e1:20:17:a9:3f:3f:fb:4d:91

br-sao.git.cloud.ibm.com:

ECDSA:

SHA256:+wLtj6TSQyY/4kp9pa8BVk+JvHqD0+L3Wq5hF3WnyCQ

MD5:c2:6a:76:c6:03:86:a0:ca:4e:24:87:9a:50:c1:3d:66

ED25519:

SHA256:sFQy3djBGVtjYuLKWtbabS95QY+yL1JQw+Wfr7eWkvw

MD5:f7:f2:39:f6:d0:a8:a6:07:9e:a6:c3:83:24:f0:7b:a3

RSA:

SHA256:3EgFqSRmbynYuCg7mpYO+K7PdSBvG0IJHr0e0Q4bikI

MD5:af:9c:57:90:25:4d:0a:ed:5c:c5:30:97:95:de:9c:92

eu-de.git.cloud.ibm.com:

DSA:

SHA256:c7Bm79CLA5y4tmnI+jB+wYp8esbIUcOSMxzHtU+hhNY

MD5:28:b7:ff:67:70:39:16:ed:fb:8a:8b:3c:26:45:b9:56

ECDSA:

SHA256:cRQsJFaZLfnQb4xOH68uZvWxuVXe0UQ9Z+ks/9dotnc

MD5:f3:02:a4:c4:63:d6:3b:30:79:fa:37:7c:ba:2c:9e:81

ED25519:

SHA256:ZVuqymHanu+N1P+OJCwHcoRlzjpvGnjV001Mo8BFEzg

MD5:84:90:72:ec:7d:ff:0e:72:01:b7:08:16:f2:76:21:87

RSA:

SHA256:33om5cGnbUduaEeKH+116IMzu2mMCHKOLTNPkmF/lNk

MD5:b3:8f:02:34:12:03:8c:41:8e:4d:be:56:1c:fe:c8:8c

eu-gb.git.cloud.ibm.com:

DSA:

SHA256:Nt0JS/AQDue0WY7X/xRC5Weu3RTplWABACiCOku8CRc

MD5:bc:a5:a2:5b:7b:c3:3d:7d:6e:d5:37:eb:08:a1:77:d3

ECDSA:

SHA256:UZPNkP+gRMINcgWSN50AeiDsOgnJGGTPXFxI/ASryag

MD5:f2:29:e2:f0:b2:33:bd:8d:19:7d:f0:41:9a:a5:f0:fe

ED25519:

SHA256:k2VN8B5ouxW+mvyp/nX3Dq7U571rluVcMx0z1iUCnU0

MD5:c6:b1:a7:4b:c7:c2:cf:38:17:32:f5:f7:8d:5b:53:a3

RSA:

SHA256:5hSoluX8hoPrChwtWZH0rEzz3Cn5bQP18cZ7xj17Wbg

MD5:e4:3b:99:ae:4b:ff:f5:f7:96:cf:cf:9a:38:3f:c7:65

jp-osa.git.cloud.ibm.com:

ECDSA:

SHA256:k+FNBh6Yvth9bWyvKnfreYhS+3s/+2MX7q2ci/tFAY0

MD5:a8:71:f1:dc:7a:28:9c:b6:fc:c6:54:1f:1c:c5:9c:08

ED25519:

SHA256:I62KQpR+VBmaJnInUj5AkStPA/Hpu555/tHBQjRjU7Q

MD5:dc:29:99:b3:4c:2a:e3:e7:b3:9b:b2:00:74:d2:b3:89

RSA:

SHA256:FPyK4sO5dzIVI/aL9Ril8GIK+uv2jiNVnTqYKDgkF24

MD5:18:4f:38:05:c8:68:61:e5:08:dc:a3:61:2d:13:45:c1

jp-tok.git.cloud.ibm.com:

DSA:

SHA256:jX4dD9ojut+OCzEtmsR6hDpK+gJ8g0B5V5k+beFzj7E

MD5:5c:62:d4:35:32:63:5a:66:79:e3:bb:be:59:ce:41:a5

ECDSA:

SHA256:ppgYQJFtPxGlx5tWLKT+aKC535C8g4Xz4Uej2BXrd1k

MD5:4c:60:e5:cd:0c:b9:3e:8e:25:dd:64:b3:7b:28:de:86

ED25519:

SHA256:xWVpw3fnjJB78HjTJOLQijjuCiQRXcPrCQ+5+rgzVLg

MD5:0a:8f:2f:55:62:9b:c5:51:ab:4b:da:e9:81:e9:02:52

RSA:

SHA256:OGttrbZoUWU5/6yjxYq9kO+VCXdQB1JkTc9shgbzrE0

MD5:c2:83:e8:3b:a8:b6:c5:da:cc:4c:26:b5:38:86:74:13

us-east.git.cloud.ibm.com:

DSA:

SHA256:onqeRZxk/GaxBVY+Bxl97UgW5rBQzTH1dJ7sGJDFUp8

MD5:d0:82:ab:e7:43:4d:92:68:70:b9:23:44:c0:5a:e3:8a

ECDSA:

SHA256:IuHvGWVB3vBJNeZ/4SRKpVgRLZHB2FbmJfU5Toek4Hk

MD5:ff:22:19:1e:83:0d:f2:bd:5d:32:84:c5:04:65:be:f6

ED25519:

SHA256:lxLtQ1Cdn5SG0ZClB9wFLSHODhJofaCUs37LdUnubNU

MD5:97:10:bd:0e:e2:e4:84:bc:fb:71:36:99:02:02:f7:66

RSA:

SHA256:TF8Pcst2F9Ek3p3cJlXz06zMwwZkoq+d23r4URtOPD8

MD5:f2:77:0c:e3:79:41:33:f5:fa:95:ce:cc:d1:dd:62:d0

us-south.git.cloud.ibm.com:

DSA:

SHA256:EX4AoOpgTqHDmZ97Klhgkz06+rSNDfe+AHZBnXzW+oc

MD5:bc:67:d0:95:80:1f:1e:c3:70:4a:66:dd:57:3b:53:d7

ECDSA:

SHA256:BQx1OpGLx8cTkoL6RmftFgTGFHBz2tKPICJm5My4fa8

MD5:2e:96:56:70:15:19:21:d6:96:d4:78:6e:84:eb:e9:d7

ED25519:

SHA256:XvuvoW6oaJjzb3BnCBrdB03B0Mbfu1Eb1/hmoLdoPDQ

MD5:e1:02:84:2c:af:d1:e7:b0:0c:6f:9c:0c:ab:c1:ec:fb

RSA:

SHA256:PEAncMcnz8jNEOmBabCtJ13cg0oGI0YxLOMWVOkDgjc

MD5:74:31:4e:57:e7:c7:12:c4:c5:96:78:f4:18:8d:63:60

You can use following code snippet to verify the host key fingerprint for a headless connection to Git over SSH, by connecting to the us-south.git.cloud.ibm.com server. To use this code for a different server, update the HOST and EXPECTED FINGERPRINT values.

HOST="us-south.git.cloud.ibm.com"

EXPECTED\_FINGERPRINT="SHA256:PEAncMcnz8jNEOmBabCtJ13cg0oGI0YxLOMWVOkDgjc"

ssh-keyscan -t rsa $HOST > /tmp/hostkey

FINGERPRINT=$(ssh-keygen -lf /tmp/hostkey | cut -d ' ' -f 2)

if [ "$EXPECTED\_FINGERPRINT" == "$FINGERPRINT" ]; then

cat /tmp/hostkey >> ~/.ssh/known\_hosts

fi

### **Updating your display name**

You can update the display name that appears for you throughout Git Repos and Issue Tracking.

1. On the [User Settings](https://us-south.git.cloud.ibm.com/profile) page, in the **Main settings** section, update your full name.
2. Click **Update profile settings** to change the name that is displayed for you throughout Git Repos and Issue Tracking.

## Physical file and repo size limits

Files are strictly limited to 100 MB. The suggested repo size limit is 1 GB. If your repo exceeds 1 GB, you might receive an email with a request to reduce the size of the repo.

### **DevOps Insights**

# Working with DevOps Insights

IBM Cloud® DevOps Insights is a tool that aggregates code, test, build, and deployment data to provide visibility of quality for all of your teams. This tutorial walks you through the quickest steps for setting up DevOps Insights with IBM Cloud® Continuous Delivery so that you can explore the features in DevOps Insights.

With DevOps Insights, you can maintain and improve the quality of your code in IBM Cloud®. You can monitor your deployments to identify risks before they are released, analyze development changes for error probability, and improve the interactions of your team.

DevOps Insights collects and analyzes the results from unit tests, functional tests, and code coverage tools. It uses these results to determine whether your code meets predefined policies at specified gates in your deployment process. If your code does not meet or exceed a policy, the deployment is halted, preventing risks from being released. You can use DevOps Insights as a safety net for your continuous delivery environment or as a way to implement and improve quality standards.

## Before you begin

Authorize the use of GitHub repos. For more information, see [Authenticating with Git Repos and Issue Tracking](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-git_working#git_authentication).

## Step 1. Add the toolchain by using a template

1. From the IBM Cloud console, click the menu icon , and select **DevOps**.
2. Expand the **Location** menu, and select a location for your deployment. DevOps Insights is available in Dallas, London, and Frankfurt.
3. Click **Create a Toolchain**.
4. Select the **Develop a Cloud Foundry app with DevOps Insights** tile.
5. In the Tool Integrations section, create an API key for Delivery Pipeline.
6. Click **Create** to finish creating the toolchain.

If you need to authorize IBM Cloud to use GitHub, click the **GitHub** tile > **Authorize**.

## Step 2. Run a build to send data to DevOps Insights

You run builds to see data within DevOps Insights. When you created this template, a build ran automatically in the Delivery Pipeline. You will see data within DevOps Insights after the build stage completes.

Click the **Delivery Pipeline** tile to view the build process. The process might take several minutes to finish. When the build stage completes, continue to step 3.

For more information about pipelines, see [Delivery Pipeline overview](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-deliverypipeline_about#deliverypipeline_about).

## Step 3. View the data analyzed by DevOps Insights

Explore the Quality Dashboard page to see the data aggregated from Delivery Pipeline. The quality dashboard provides quality data sets for each application.

1. Click the menu icon , and select **DevOps**.
2. On the Toolchains page, click the DevOps Insights toolchain to open its Overview page.
3. On the **IBM Cloud tools** card, click the DevOps Insights tool integration.
4. Click **Quality Dashboard**.

You can view details about the Weather Application and the quality of the code that was analyzed. These tests are available where the policy gates passed: code coverage, unit test, and the functional verification test. You can click the build ID, for example, master:1, to view a summary for that specific test. Click **View trends** to view trend details.

For more information about the quality dashboard, see [DevOps data aggregation](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-devops-data-aggregation).

## Step 4. Test a gate against a DevOps Insights policy decision

A gate is created when you enact a policy. A policy is a set of rules that you can customize, and a rule is the passing criteria that you define for each type of test data you upload. The gate passes or fails a build based on the quality standards that you choose. So, if your code doesn't meet or exceed a policy that is enacted at a particular gate, the deployment is stopped to prevent risky changes from being released.

For the Weather Application, the code coverage rule set for the policy is that the minimum code coverage required is 80%. The gate is placed before the production stage within the pipeline. When you first create this template, the app passes all current gates, but gates sometimes fail.

To make a gate fail, edit the code in the routes/apivl.js file so that the code coverage reaches only 60%. The gate fails because the code doesn't reach the necessary quality and deployment isn't pushed to production.

1. From your toolchain, click the **Eclipse Orion Web IDE** tile.
2. Open the routes/apivl.js file, and uncomment lines 42-72.
3. Save the file by clicking **File**, and select **Save**.
4. Select the Git icon, enter a commit message, and click **Commit**.
5. Click **Push** to push your changes.
6. Click the back arrow in the Eclipse editor to return to your toolchain.
7. On the toolchain's Overview page, on the **IBM Cloud tools** card, click the DevOps Insights tool integration to observe the gate fail in real time.

## Step 5. Analyze the failed gate

The gate fails because the code coverage isn't met. When you gate your own deployments, you can determine whether it failed or passed by looking at the Risk Analysis page. Also, you can define, change, and customize policies and rules to fit your needs when it comes to gating deployments. View the policies and rules that make the gate.

1. Click the menu icon , and select **DevOps**.
2. On the Toolchains page, click the DevOps Insights toolchain to open its Overview page.
3. On the **IBM Cloud tools** card, click the DevOps Insights tool integration.
4. Click **Policies** > **Weather Unit Test, Code Coverage, and FVT Checks**.
5. Click **Code coverage** to view the minimum code coverage required. Anything equal to or over 80% will release to the next stage.
6. Click **Risk analysis** to check whether your deployment passed or failed the gate. Risk is evaluated based on the defined policies within DevOps Insights.
7. Select the build with the failed policy to view the test summary details.

## Alternative tutorials

This tutorial focuses on implementing DevOps Insights with IBM® Continuous Delivery Pipeline for IBM Cloud®, but as an alternative, you can use DevOps Insights with Jenkins and other CI/CD tools. Use the following tutorials for more information.

* [Integrate DevOps Insights with an IBM Continuous Delivery pipeline](https://www.ibm.com/cloud/garage/tutorials/integrate-devops-insights-with-cd-pipeline). Learn how to configure a CD pipeline to send, build, and deploy information to DevOps Insights and define policies that analyze deployment risk.
* [Integrate DevOps Insights with Jenkins](https://www.ibm.com/cloud/garage/tutorials/use-jenkins-plugin-to-post-data-to-devops-insights). Learn how to set up and use the DevOps Insights Jenkins plug-in to publish build, test, and deployment data to DevOps Insights.
* [Integrate DevOps Insights using the IBM Cloud CLI](https://www.ibm.com/cloud/garage/tutorials/use-cli-to-post-data-to-devops-insights). Learn how to set up the environment and use the CLI to publish build, test, and deployment data to DevOps Insights.

### **Orion Web IDE**

# Developing with the Eclipse Orion Web IDE

The Eclipse Orion Web IDE is a browser-based development environment where you can develop for the web in JavaScript, HTML, and CSS with the help of content assist, code completion, and error checking. The Web IDE works with nearly any language and you can highlight syntax for most file types. Source control is built in, and you can deploy code locally to test and debug your apps.

Best of all, the Web IDE is powered by the web. You have nothing to install, nothing to maintain, and nothing to scale. You can develop anywhere that you have an internet connection.

Don't store regulated data in files within the Web IDE. The procedures for regulated data are currently not in place.

## Setting up the IDE

The Web IDE is customizable so that you can choose the color schemes, technical tools, and settings that meet your development needs. To view and modify the settings, from the navigation sidebar on the left, click the **Settings** icon Settings icon.

If you often need to change certain settings while you edit, you can access those settings quickly from the **Local Editor Settings** icon Local Editor Settings icon.

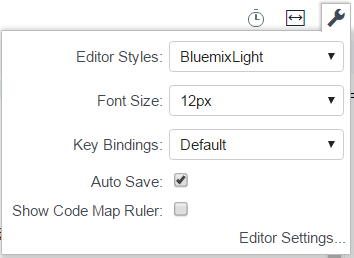
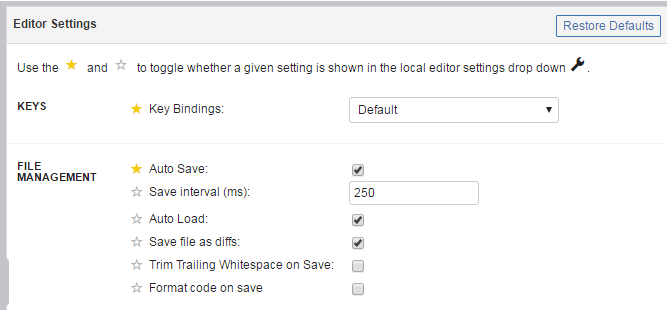


Figure 1. Local Editor Settings

By default, the settings for the editor style and font size are always shown. To include other editor settings in the menu, follow these steps:

1. Click the **Local Editor Settings** icon Local Editor Settings icon.
2. Click **Editor Settings**.
3. To include or exclude a setting from the **Local Editor Settings** menu, click the star for each setting.

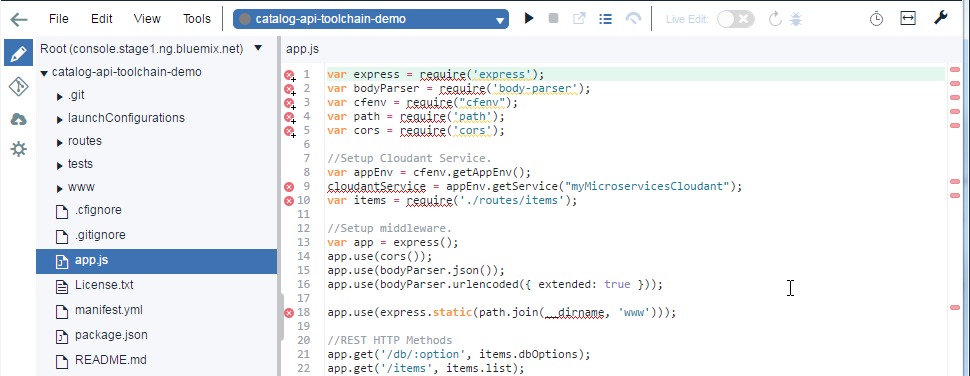
Figure 2. Local Editor settings

## Editing code

The Web IDE has two main sections. The first section is the file navigator, which shows your project files in a tree structure. From the file navigator, you can create, rename, delete, and manage your files and folders.

To upload files to the file navigator, drag them from your computer to the file navigator.

The second section is the editor pane. The editor provides several coding features, including content assist and syntax validation.

Figure 3. Web IDE

### **Working with multiple files**

1. To work with two files at the same time, click the **Change split editor mode** icon Split Editor icon.
2. From the menu that opens, select a view.

After you select a view, if a file was already open in the editor, it is shown in both editor views.

To open or change a file that is shown in one of the editor views:

1. Move the cursor to the editor view that you want to change.
2. In the file navigator, click a file.

### **Keyboard shortcuts**

Many of the commands in the Web IDE are accessible through keyboard shortcuts.

To see a list of the keyboard shortcuts in the editor, click **Tools** > **Show keys**. Alternatively, you can see the list by pressing Alt+Shift+?, or on MacOS, Ctrl+Shift+?. You can customize a shortcut by hovering over the key, clicking the pencil, and typing the new key binding.

## Managing source code

The Web IDE is integrated with source code management tools. To work with your Git repository, click the **Git Repository** icon Git Repository icon. For more information, see [Working with Git in the Eclipse Orion Web IDE](https://cloud.ibm.com/docs/services/ContinuousDelivery?topic=ContinuousDelivery-git_web_ide#git_web_ide).

## Deploying an app from your workspace

1. To deploy your app, from the run bar, either select or create a launch configuration.

Run barFigure 4. Run bar

1. Click the deploy icon deploy icon. An instance of your app is deployed by using the current contents of your workspace and the environment that is defined in your launch configuration.
2. After your app is deployed, you can use the run bar to stop, restart, or debug your app, view logs, and more.

| **Run Bar Icon** | **Description** |
| --- | --- |
| Stop icon | Stop the app. |
| Open app URL icon | Open the deployed app. |
| View logs icon | View the logs of the deployed app. |
| Open dashboard icon | Open the app's dashboard. |

If you are developing a Node.js app, enable Live Edit mode to restart the app and access the debugger: The enable live edit slider

| **Run Bar Icon** | **Description** |
| --- | --- |
| The Live Edit restart icon | With Live Edit mode enabled, restart the app quickly, without redeployment. |
| The debug icon | With Live Edit mode enabled, access the debugger. |

## Supported languages

The Eclipse Orion Web IDE provides content assist, tooltips, previews, validation, and highlights syntax for JavaScript, HTML, CSS, and Markdown files. You can also highlight syntax for these file types:

* Arduino
* C#
* C++
* CoffeeScript
* CSHTML
* Embedded JavaScript (ejs)
* Erlang
* Go
* HTML abstraction markup language (Haml)
* Jade
* Java
* JSON
* Less
* Lua
* Objective-C
* PHP
* Python
* Ruby
* Sass/SCSS
* SQL
* Swift
* TypeScript
* Visual Basic (vb)
* VMHTML
* XHTML
* XML
* XQuery
* YAML
* Launch file
* Dockerfile
* gitignore
* git config
* cfignore
* properties

Reference:

<https://cloud.ibm.com/docs/apps?topic=apps-devops-toolchains>

<https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains_getting_started&interface=ui>

<https://www.ibm.com/cloud/architecture/toolchains/>

<https://www.ibm.com/cloud/architecture/tutorials/introduce-develop-cloud-foundry-app-toolchain/>

<https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-devops_intro>