Learning Outcome

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

1. Able to Build a web application on modern cloud-based architectures and services.
2. IBM Cloud DevOps Toolchain & Services
3. Serverless Architectures
4. What is the cloud provider’s role in serverless?
5. Why use serverless architectures?
6. How does Serverless Architectures Work?
7. Function-as-a-Service (FaaS)
8. Popular Serverless Platforms
9. Serverless Architecture Use Cases
10. Advantages of using serverless architecture
11. Limitations serverless architecture

# Migrating application to Cloud

## Application Migration

**What is application migration?**

Application migration is the process of moving a software application from one computing environment to another. You might, for instance, migrate an application from one data center to another, from an on-premises server to a cloud provider’s environment, or from the public cloud to a private cloud environment.

Because applications are typically built to run on particular operating systems in specific network architectures or developed for a single cloud platform, moving an application to a new environment can pose a number of challenges. It’s usually easier to migrate applications from virtualized or service-based architectures than it is to migrate those running on bare metal hardware.

Determining an overall application migration strategy involves considering each individual application’s dependencies and technical requirements, as well as your enterprise’s security, compliance, and cost constraints.

Different applications can take different paths to the cloud, even within the same technology environment. Since the early days of cloud computing, developers have referred to these application migration patterns with names that begin with “R."

## Cloud Migration Strategy

There’s a well-known framework for organizing your strategies for cloud migration: this is known as “The 6 Rs of Cloud Migration”. Not every business will perform each step but think of this as a guide to illustrate the many possible paths an organization can take. Once you review their details, you’ll have more context to understand which way to lead your migration strategy.



Image: Cloud Migration Strategy

Reference: <https://i0.wp.com/v3it.com/blog/wp-content/uploads/2021/07/cloud-migration-types.png?resize=605%2C473&ssl=1>

1. **Rehost:** Rehosting is often called “lift and shift”. Just as the name implies, there is no big architectural change to the servers and applications in this situation. They are simply taken from on-premises (the lift) and moved to the same type of system on the cloud (the shift). Organizations that are just starting their migration journey will often use the lift-and-shift strategy.

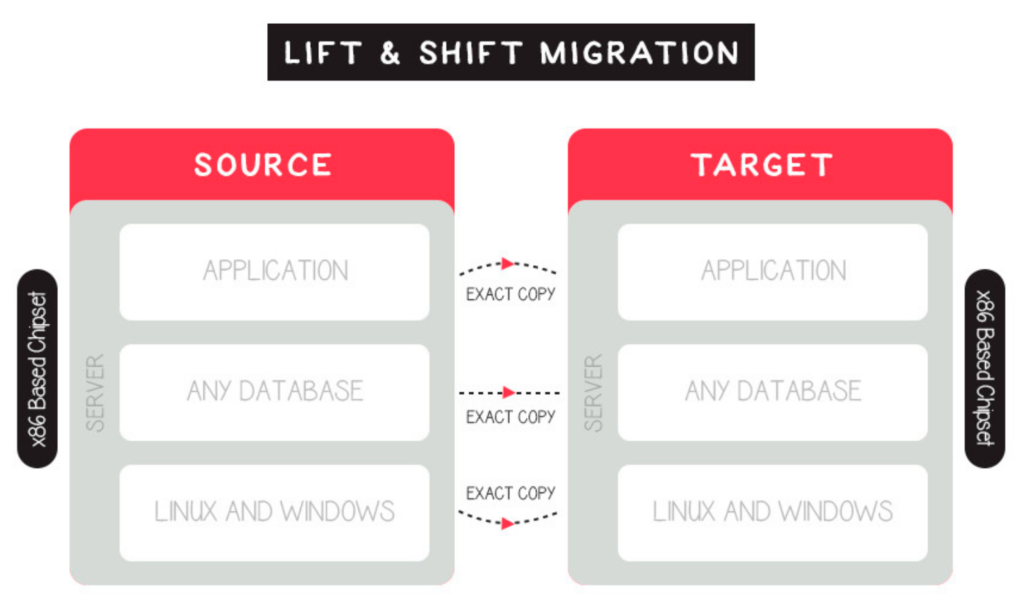


Image: Lift and shift cloud migration

Reference: <https://cloudacademy.com/wp-content/uploads/2021/12/Lift-and-shift-cloud-migration-1024x600.png>

1. **Replatform:** Replatforming is the second option. This is where we modify “lift and shift” into something more complicated but better suited to the new cloud environment. Replatforming is a process that optimizes the application during the migration phase. This requires some programming knowledge and input. You might move from your own database system to a managed DB hosted on a cloud provider. In this type of migration, you stick with similar underlying technology but modify the business model and have cloud resilience as a huge bonus.
2. **Repurchase:** Sometimes referred to as “drop and shop,” this cloud migration strategy comprises a full switch to another product. This could mean ending existing licensing or repurposing services for new platforms and services. In this instance, some examples of a “dropped” application may be a CRM system or an industry-specific app that was not created to be run on the cloud. However, it may be one that does not have modern code or one that cannot be transported from one provider to the next. When transferring to a new product or using a proprietary platform, the “repurpose” strategy is used.
3. **Refactor:** Refactoring is the fourth R, which is basically redesigning. This is often driven by a want to improve an application or service. This could be due to various factors such as difficulty in improving the environment or the need to increase the availability and reliability of an application to meet anticipated traffic spikes.The timing of refactoring is important. While it may be possible to re-architect the application during the migration stage if the application is not mission-critical, It’s generally best to do this later in the project. It’s important to remember that refactoring can take some time and requires expertise.

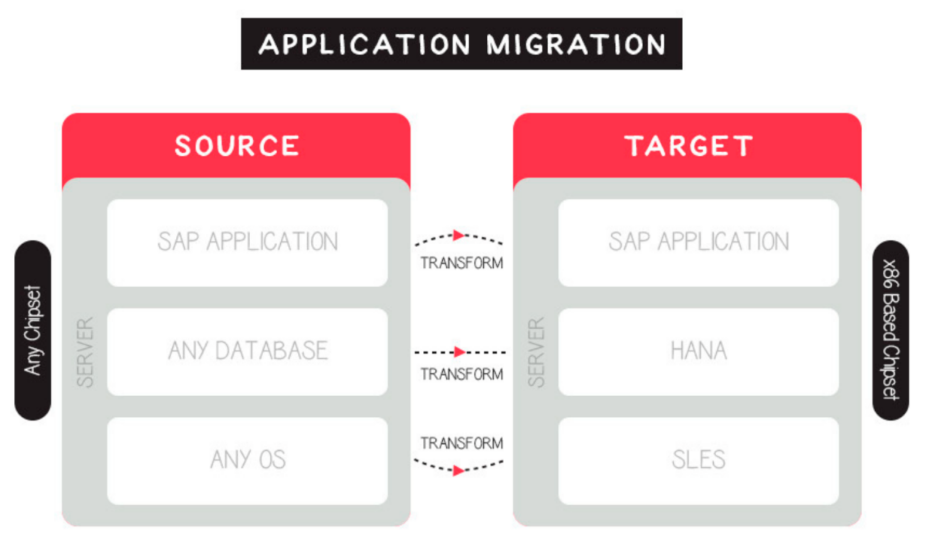


Image: Refactoring type of cloud migration

Reference: <https://cloudacademy.com/wp-content/uploads/2021/12/Refactoring-type-of-cloud-migration-1200x702.png>

1. **Retain:** Retain is the fifth strategy. Some applications may be too difficult to migrate, so you might want to keep them. This is when you jump into the hybrid space, like many other successful enterprises. There can be various reasons why you might want to keep some of your existing on-premises deployments: if you are currently subject to regulations or have rules regarding the storage or operation of certain aspects of your business applications, services, or data on-premises or in specific areas, this approach may be a good option.
2. **Retire:** We now have our final strategy: to retire services. This strategy involves identifying assets that can be retired so that the business can concentrate on services that are most used and have immediate value. This is an interesting way to approach your existing application library because even though there may be big changes to be made, you can see them as opportunities.

## Benefits of Migrating to the Cloud

Here are some of the benefits that compel organizations to migrate resources to the public cloud:



Image: Cloud Migration Benifits

Reference: <https://www.i2econsulting.com/wp-content/uploads/2020/04/Azure-2-1.jpg>

1. **Scalability -** cloud computing can scale to support larger workloads and more users, much more easily than on-premises infrastructure. In traditional IT environments, companies had to purchase and set up physical servers, software licenses, storage and network equipment to scale up business services.
2. **Cost -** cloud providers take over maintenance and upgrades, companies migrating to the cloud can spend significantly less on IT operations. They can devote more resources to innovation - developing new products or improving existing products.
3. **Performance -** migrating to the cloud can improve performance and end-user experience. Applications and websites hosted in the cloud can easily scale to serve more users or higher throughput, and can run in geographical locations near to end-users, to reduce network latency.
4. **Digital experience -** users can access cloud services and data from anywhere, whether they are employees or customers. This contributes to digital transformation, enables an improved experience for customers, and provides employees with modern, flexible tools.

## Application migration risks

Stakeholders may fear that application migrations might cause disruptions to the business or result in unanticipated costs. The most common risks include the following:

1. **Unforeseen technical challenges:** For example, an application may have so many dependencies that refactoring or Replatforming can be much more complex and time-consuming than originally thought.
2. **Unanticipated costs:** Without proper planning, businesses may incur expenses that they hadn’t budgeted for, such as new licensing fees or the training costs associated with getting employees up to speed on new tools.
3. **Unexpected downtime:** Major changes to an application can cause conflicts or issues that lead to unplanned downtime, both for the application and for connected or dependent systems.
4. **Cultural issues or change management difficulties:** Different organizations use apps differently, and those differences can create friction that slows down a migration project.

Undertaking a careful and detailed assessment of the risks and benefits associated with rehosting, re-architecting/Replatforming, or retiring each application in your portfolio can help mitigate the overall risks associated with application migration. In particular, it’s important to compare department-level costs with the total cost to the enterprise and to assess the total cost of ownership (TCO) of any hardware you’d need to maintain to keep applications on premises.

## Application migration plan in three stages

Generally speaking, the application migration planning process can be divided into three stages. In each, it’s critical to weigh the costs of all potential options, including choosing to retain some on-premises workloads.

1. **Application identification and assessment**

In this initial discovery phase, you should begin by ensuring you have a comprehensive catalog of all applications in your portfolio. You’ll then categorize the applications according to whether they have business-critical or non-critical importance, whether their value is strategic or non-strategic, and what you stand to from migrating each to the cloud. You should strive to understand each application’s value in terms these characteristics:

* Impact on the business
* Ability to fulfill crucial business needs
* Timeliness and importance of data
* Size, complexity, and manageability
* Cost of maintenance and development
* Increased value from migration to the cloud

You’ll then want to conduct a cloud affinity assessment for each application you’re thinking about migrating. During this process, you can determine which applications are ready to go as-is and which would need significant changes before they could be made cloud-ready.

You can also employ application dependency discovery tools to help you determine the feasibility of migrating a particular workload outside its current environment.

1. **Total cost of ownership (TCO) assessment**

Determining the total cost of a cloud migration project can be a complex undertaking. You’ll need to compare “what-if” scenarios for keeping applications and infrastructure on-premises with those associated with moving them to the cloud. This means you’ll have to calculate purchasing, operating, and maintenance costs for the hardware you’d maintain on-premises in either scenario and the costs of licensing software.

You’ll want to compare the monthly bill you’d get from your cloud provider in either scenario and the costs of the migration itself, including the costs of testing the new infrastructure and training employees to use updated software. Don’t forget to consider maintenance costs for legacy applications that remain on-premises.

1. **Assess overall risk and project duration**

In the final phase of migration planning, you’ll establish a timeline for the project and identify any risks or stumbling blocks that you are particularly likely to encounter.

**Legacy application migration to cloud**

Generally speaking, the older the application, the more challenging (and as a result, potentially the less worthwhile) it is to migrate to the cloud. Outdated software is problematic in many ways: it’s expensive to maintain, it can raise security concerns if it’s no longer being patched, and it tends to perform poorly in modern computing environments. Be especially thorough with your assessment of legacy applications before deciding to migrate them.

## Cloud migration checklist

To ease your transition to the cloud, prepare a checklist that helps keep the project on track by checking off each task as completed. A checklist can be as basic or exhaustive as project managers choose to make it. The following are some items to include:

* Determine which workloads will be relocated to the cloud and classify them by complexity, size, and production/not production.
* Research and select a cloud provider suitable to the workloads being relocated.
* Determine if you will need a multicloud approach based on your workloads.
* Perform a cost assessment for the migration.
* Assign a team to execute the migration.
* Communicate the goals of the migration to the team.
* Determine how much of the migration will be handled internally and by the cloud provider.
* Prioritize which workloads to migrate first.
* Prepare a plan outlining the roadmap and schedule for the migration.
* Ascertain whether the organization already uses any cloud-based applications and whether they should remain as they are or be replaced by new cloud-based services.
* Communicate to all stakeholders what to expect during and post-migration.
* Prepare a security plan for migration and post-migration.
* Establish KPIs for the migration.
* Check in with implementers along the way to review progress.
* Test, review, and make adjustments as needed.

## Application migration and IBM Cloud

IBM Cloud offers a full suite of cloud migration tools and services, ranging from IBM Assisted Migration Services, which include fully-managed migrations using the customer’s own tools, to seamless lift-and-shift migrations of VMware workloads supported through IBM’s partnership with VMware. (You can even calculate the value of migrating with IBM Cloud for VMware solutions.)

**Make you deploy-anywhere future possible**

Unlock your core applications and harness cloud-native development to become a nimble market competitor.

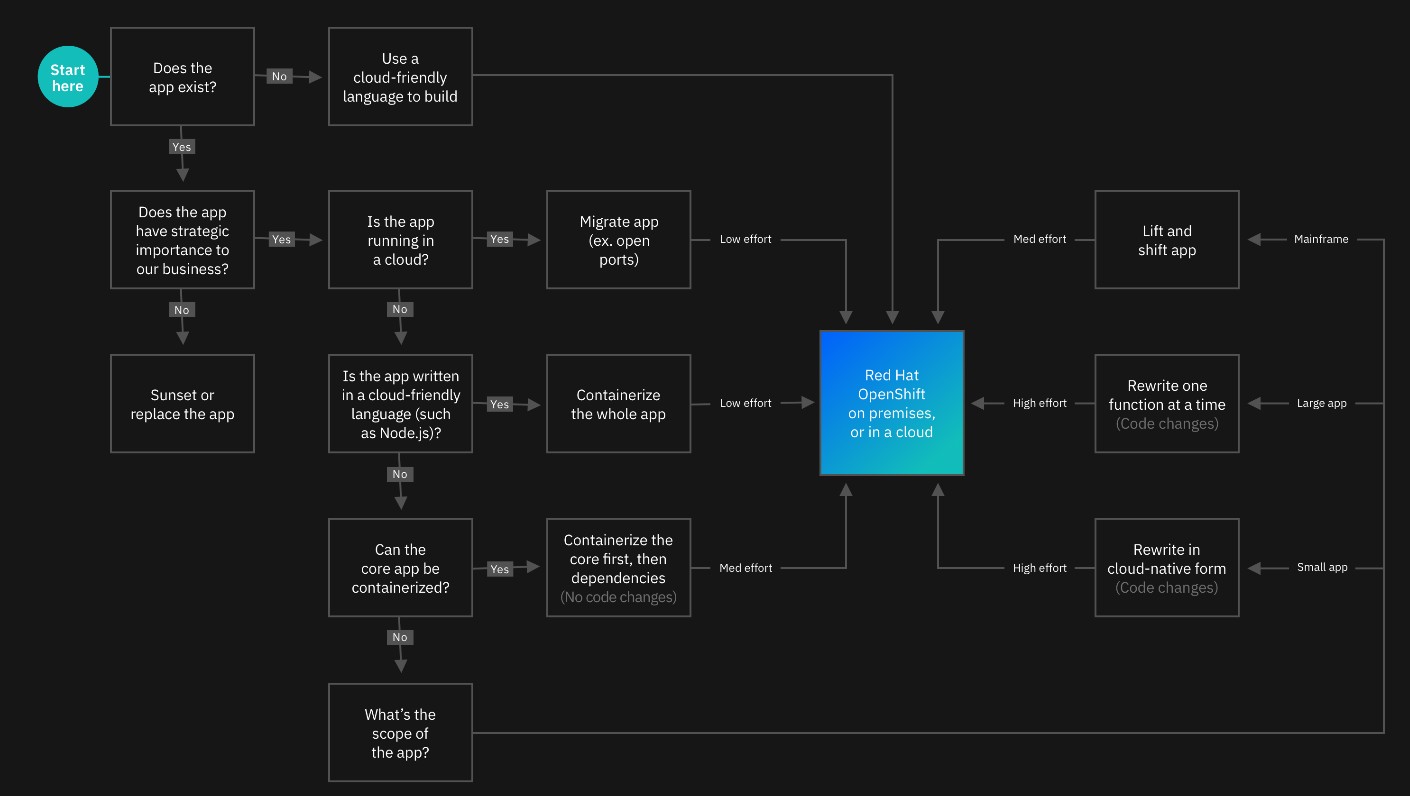


Image 1: Cloud Migration

### **Introduction**

**Tackle modernization and cloudification with IBM**

When faced with upgrading your application estate, you run into a lot of questions. Which technologies do you adopt? How can you reduce technical debt? What apps do you build new in the cloud? How can you be sure you’ll keep pace with industry changes? And how do you do it all fast? Working with IBM, you’ll have an experienced partner that can guide you.

### **IBM approach**

**Be part of the process with an end-to-end partner**

IBM works with you through your entire application transformation—from the modernization of existing infrastructure and applications to the development and deployment of applications that run on any cloud platform. Together, we’ll start small, develop an ROI-based strategy aligned to your goals and harness quick wins. Then we’ll show you how to begin to change your culture, scale across your business and make your journey to the cloud a success. Using DevOps tools, accelerators and AI-driven automation, we’ve helped clients change how they operate. We'll do the same for you.

### **Strategy**

**Develop a strategy that delivers**

Do you need to optimize and modernize? Should you rehost, replatform or redesign? Through thousands of client engagements across 20 industries, we’ve learned what works and what doesn’t. Each application estate is unique and requires a custom solution. Our assessment starts with AI- driven insights and visualizations of your applications, middleware and databases. You’ll come away with a complete ROI-driven roadmap. From there, you can implement it yourself using our software, IBM can implement it for you or IBM can manage all of it for you on an ongoing basis.

### **Modernization**

**Reduce application costs**

Yes, you can reduce your technical debt. According to a recent Forrester TEI study, IBM has helped clients optimize applications and migrate to the cloud fast, with measurable impact, including reduced compliance risk and migration cost, and improved deployment speed. Our skills, methods and tooling also go beyond modernization, helping you align your IT with business requirements, reduce your operating costs to free up innovation capital and enhance your overall application security.

### **Cloud native**

**Adopt modern dev tools and methods**

IBM provides environments, tools and methods to help you deliver applications that run anywhere

— on premises or on any cloud — faster and securely. You’ll be able to build containerized applications, optimized for scale and performance, that take advantage of continuous delivery and DevSecOps principles to reduce time to market.

### **Integration**

**Ensure access to data and application services**

Traditional integration architectures can’t collect the kinds of actionable business insights that come from real-time access to data and applications. To do this, they must modernize. IBM helps you build an architecture that’s cloud-native in design, API-led and event-driven — with fine- grain deployment through containers. This modern integration architecture empowers your teams to build the integrations they need to meet demands while lowering costs.

### **Deployment**

**Quickly deploy and manage apps at scale across clouds**

Wherever your workloads run, we can help you deploy and manage your applications securely across environments, clouds, including IBM Cloud and specialized hardware. Some problems may benefit from the portability that comes from containerization and Red Hat® OpenShift®. Other problems are better solved with a single control plane across clouds that has built-in support for security and compliance management. In other cases, it might make sense to have IBM manage your estate for you. Regardless of the approach, IBM can help you reduce costs while increasing operational efficiency and visibility.

### **Culture change**

**Evolve your culture with IBM Garage**

In IBM Garage™, we work side-by-side with you, even if it that means virtual, to uncover new ways of working that help you drive transformational change. We bring together the depth of experience and proven methodology of IBM and use enterprise design thinking to draw on your team’s wisdom and creativity. With the right people, useful data, applied technology and dedicated spaces, we can co-create with you to build cloud-native applications in as little as six to eight weeks.

# Understanding DevOps, tools, DevOps services in Cloud

## DevOps

DevOps speeds delivery of higher quality software by combining and automating the work of software development and IT operations teams.

Let’s go to the good old days. Once in a time again, there are two separate teams, developers and operators. Developers develop an application. And, operators deploy this application to servers and maintain it on a server. These days, everyone was happy. After the application is developed and deployed, there is no need to update much. It would take too long to adapt to changes and make updates on our software. As a result of this, developers and operators are two different teams. But time changes like everything. The change is inevitable. In our modern times, we witness those applications are constantly updated. So, it leads the IT companies to a new culture, called DevOps.

### What is DevOps?

DevOps (a portmanteau of “development” and “operations”) is the combination of practices and tools designed to increase an organization’s ability to deliver applications and services faster than traditional software development processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

In simple terms, DevOps is about removing the barriers between traditionally siloed teams, development and operations. Under a DevOps model, development and operations teams work together across the entire software application life cycle, from development and test through deployment to operations.

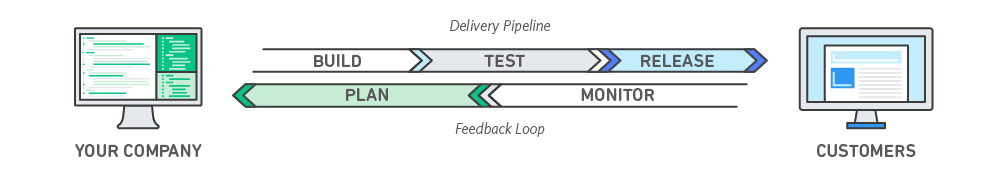


Image 1: DevOps

Reference: <https://aws.amazon.com/devops/what-is-devops/>

### DevOps Lifecycle

DevOps focuses on bringing all the development, operations, and IT infrastructure guys, including Developers, Testers, System Admins, and QAs, under one roof. Hence, all these people together are called DevOps Engineers.

DevOps Engineers share the end-to-end responsibility of gathering information, setting up the infrastructure, developing, testing, deploying, continuous monitoring, and fetching feedback from end-users. This process of developing, testing, deploying, and monitoring keeps on repeating for better results.

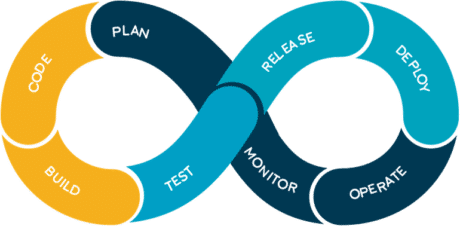


Image 2: DevOps Lifecycle

Reference: <https://aws.amazon.com/devops/what-is-devops/>

**Code:**

The first step in the DevOps life cycle is coding, where developers build the code on any platform

**Build:**

Developers build the version of their program in any extension depending upon the language they are using

**Test:**

For DevOps to be successful, the testing process must be automated using any automation tool like Selenium

**Release:**

A process for managing, planning, scheduling, and controlling the build in different environments after testing and before deployment

**Deploy:**

This phase gets all artifacts/code files of the application ready and deploys/executes them on the server

**Operate:**

The application is run after its deployment, where clients use it in real-world scenarios.

**Monitor:**

This phase helps in providing crucial information that basically helps ensure service uptime and optimal performance

**Plan:**

The planning stage gathers information from the monitoring stage and, as per feedback, implements the changes for better performance

### Different Lifecycle Stages



Image 3: Stages of DevOps Lifecycle

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

The DevOps lifecycle provides a structure to the project in such a way that it gives the team or the person working on it a view of what comes next. By following this approach, one can develop a quality project in very little time and with high reliability. One cannot simply say that he/she knows DevOps without knowing its lifecycle. Here are the various stages of the DevOps lifecycle

* Continuous Development
* Continuous Integration
* Continuous Testing
* Continuous Monitoring
* Continuous Feedback
* Continuous Deployment
* Continuous Operations

**Continuous Development**

The continuous development phase involves planning and coding the product the team is developing. In this phase of the DevOps lifecycle, the vision and goal of the project are set, and developers start to code. The integration of development and operations teams helps in planning the work accordingly, increasing the productivity of the team. **In this phase, they use tools, such as Git, CVS, Slack, etc.**

Before DevOps, the concept of the cloud was in just its initial stages, and companies had to use fixed hardware and software allocations they had planned for the project. Now, with cloud services in place, they can plan to increase or decrease the resource allocation for the project using cloud resources within their budget.

With the adaptation of DevOps, there occurred an increase in the usage of good coding methodologies and versioning systems. Take Git, for example. Using Git and its commands, users can maintain version control for keeping track of the changes made to a set of files so that, whenever the newer versions have serious bugs or critical vulnerabilities in them, the team can revert to previous versions.

Similarly, Slack, Skype, and more recently Zoom are used for communication between the team members where they can send messages directly or hold virtual meetings to keep track of the progress of the project.



Image 4: Continuous Development

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**Continuous Integration**

In the continuous integration phase, the source code in the central repository is regularly updated by developers. This phase not only involves compilation but also unit testing, integration testing, code review, and the packaging of the code written by the developers. **The tools used in this DevOps process are Jenkins, GitLab, etc.**

Jenkins orchestrates a chain of actions that helps it achieve the continuous integration process in an automated fashion. It is a server-based application and uses servers like Apache Tomcat. The reason why it is used so much is it monitors the repeated tasks that arise during the development of a project and continually tests the builds to show errors, if any, in the early stages of development itself.



Image 5: Continuous Integration

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**Continuous Testing**

In the continuous testing phase, the code written by developers is sent to testers where they use automated tools to test it for bugs. The beauty of this phase is that they can schedule to run the tests automatically at a predefined time. The report generated in this phase is sent back to the developers where they make necessary updates to the code to remove the bugs.

**The tools used in this DevOps process are JUnit (to test the Java code), Selenium, and Docker to simulate a test environment in a container so that the rest of the code is not disturbed.**

Selenium is an automated testing framework used to validate applications across various browsers and platforms. You can create Selenium test cases using various programming languages, such as Java, Python, C#, etc. It is not just a single tool but a suite of software where each piece is used for the different QA testing needs of an organization.

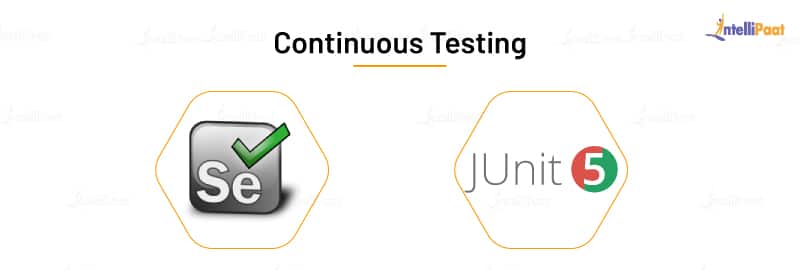


Image 6: Continuous Testing

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**Continuous Monitoring**

The continuous monitoring phase involves monitoring the health, performance, and reliability of the application or code, as well as the infrastructure, as the phases move from development to deployment. **The tools used in this phase are Nagios, Sensu, etc.**

Nagios is a platform that tracks the infrastructure, networks, and systems. It monitors and alerts services for servers, switches, software, etc. If there is a problem, it warns the users and notifies them again when the problem gets solved.

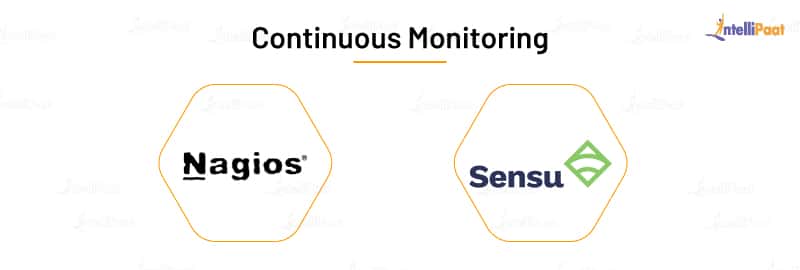


Image 7: Continuous Monitoring

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**Continuous Feedback**

In the continuous feedback phase of the DevOps lifecycle, the evaluation of the effect of each release on the user experience takes place, and this evaluation is reported back to the team to improve the future releases.

The feedback can be gathered in two methods: structured and unstructured. The structured method is applied through surveys, questionnaires, and focus groups. The unstructured feedback collection is done through social media platforms such as Twitter, Facebook, etc. Here, the users take part in this DevOps process by providing their feedback, just like how users provide app reviews on Google Playstore.

In this phase, the team uses **Pendo,** which is a product-analytics tool that helps organizations get customer views. It gives user insight, user guidance, user sentiment, and user feedback to an organization to know what its users want or what they are expecting.



Image 8: Continuous Feedback

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**Continuous Deployment**

In the continuous deployment phase of the DevOps lifecycle, the application is deployed on the production server to make it available for the intended users. **The tools used in this phase are AWS CodeDeploy, Octopus Deploy, Jenkins, etc.**

AWS CodeDeploy is a software deployment service that automates deployment to a variety of services. It makes it easier for organizations to rapidly release new features, avoiding downtime during the deployment, and it handles the complexity of the deployment process. It automates software deployment, eliminating the need for error-prone manual operations. It also scales the resources to match deployment needs.

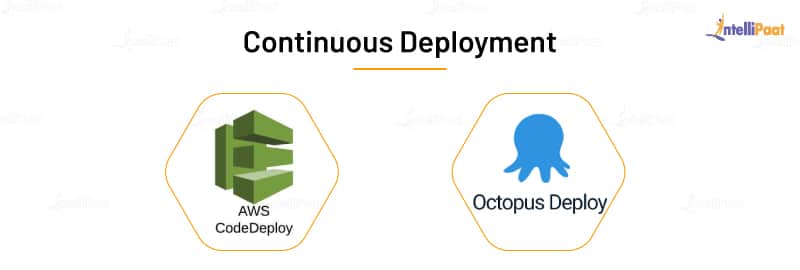


Image 9: Continuous Deployment

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**Continuous Operations**

The continuous operations phase involves the reduction or elimination of planned downtime like scheduled maintenance. The goal of this phase is to increase the uptime or the time the users can use the application. Companies use container management systems like Kubernetes or Swarm in this phase.

When developers want to make updates to the production server, usually they have to take it offline and make changes to it. This would increase the downtime of the software bringing loss to the organization.

To decrease and eliminate that downtime, they can use Kubernetes. They take a container with the software managed by Kubernetes and make the necessary changes to it, while Kubernetes runs another container containing the current version of the software. When the team deploys the software with changes, **Kubernetes** make those changes to all the containers present in the server without the team manually doing it.

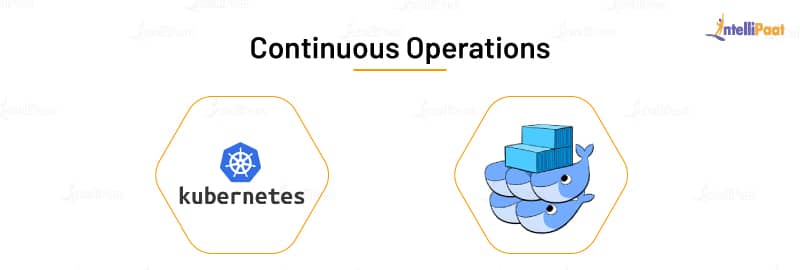


Image 10: Continuous Operations

Reference: <https://intellipaat.com/blog/what-is-devops-lifecycle-and-process/>

**DevOps lifecycle in IBMCloud**

The DevOps lifecycle (sometimes called the continuous delivery pipeline, when portrayed in a linear fashion) is a series of iterative, automated development processes, or workflows, executed within a larger, automated and iterative development lifecycle designed to optimize the rapid delivery of high-quality software. The name and number of workflows can differ depending on whom you ask, but they typically boil down to these six:

* Planning
* Development
* Integration (or build, or continuous Integration and continuous delivery (CI/CD)
* Deployment (usually called continuous deployment)
* Operations
* Learning



Image 11: DevOps lifecycle in Ibmcloud

Reference: <https://www.ibm.com/cloud/learn/devops-a-complete-guide>

**Planning (or ideation):** In this workflow, teams scope out new features and functionality in the next release, drawing from prioritized end-user feedback and case studies, as well as inputs from all internal stakeholders. The goal in the planning stage is to maximize the business value of the product by producing a backlog of features that when delivered produce a desired outcome that has value.

**Development:** This is the programming step, where developers test, code, and build new and enhanced features, based on user stories and work items in the backlog. A combination of practices such as test-driven development (TDD), pair programming, and peer code reviews, among others are common. Developers often use their local workstations to perform the “inner loop” of writing and testing code before sending it down the continuous delivery pipeline.

**Integration (or build, or continuous Integration and continuous delivery (CI/CD):** As noted above, in this workflow the new code is integrated into the existing code base, then tested and packaged into an executable for deployment. Common automation activities include merging code changes into a “master” copy, checking out that code from a source code repository, and automating the compile, unit test and packaging into an executable. Best practice is to store the output of the CI phase in a binary repository, for the next phase.

**Deployment (usually called continuous deployment):** Here the runtime build output (from integration) is deployed to a runtime environment - usually a development environment where runtime tests are executed for quality, compliance and security. If errors or defects are found, developers have a chance to intercept and remediate any problems before any end users see them. There are typically environments for development, test, and production, with each environment requiring progressively “stricter” quality gates. A good practice for deployment to a production environment is typically to deploy first to a subset of end users, and then eventually to all users once stability is established.

**Operations:** If getting features delivered to a production environment is characterized as “Day 1”, then once features are running in production “Day 2” operations occur. Monitoring feature performance, behavior, and availability ensures that the features are able to provide value add to end users. Operations ensures that features are running smoothly and that there are no interruptions in service - by making sure the network, storage, platform, compute and security posture are all healthy! If something goes wrong, operations ensure incidents are identified, the proper personnel are alerted, problems are determined, and fixes are applied.

**Learning (sometimes called continuous feedback):** This is the gathering of feedback from end users and customers on features, functionality, performance and business value to take back to planning for enhancements and features the next release. This would also include any learning and backlog items from the operations activities, that could empower developers to proactively avoid any past incidents that could happen again in the future. This is the point where the “wraparound” to the Planning phase happens and we “continuously improve!”

Three other important continuous workflows occur between these workflows:

**Continuous testing:** Classical DevOps lifecycles include a discrete “test” phase that occurs between integration and deployment. However, DevOps has advanced such that certain elements of testing can occur in planning (behavior-driven development), development (unit testing, contract testing), integration (static code scans, CVE scans, linting), deployment (smoke testing, penetration testing, configuration testing), operations (chaos testing, compliance testing), and learning (A/B testing). Testing is a powerful form of risk and vulnerability identification and provides an opportunity for IT to accept, mitigate, or remediate risks.

**Security:** While waterfall methodologies and agile implementations 'tack on' security workflows after delivery or deployment, DevOps strives to incorporate security from the start (Planning) - when security issues are easiest and least expensive to address - and continuously throughout the rest of the development cycle. This approach to security is referred to as shifting left.

**Compliance:** Regulatory compliance (governance and risk) are also best addressed early and throughout the development lifecycle. Regulated industries are often mandated to provide a certain level of observability, traceability and access of how features are delivered and managed in their runtime operational environment. This requires planning, development, testing, and enforcement of policies in the continuous delivery pipeline and in the runtime environment. Auditability of compliance measures is extremely important for proving compliance to 3rd party auditors.

### How DevOps works?

Under a DevOps model, development and operations teams are no longer “siloed.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team’s velocity.



Image 12: How DevOps works?

Reference: <https://www.romexsoft.com/wp-content/uploads/2019/09/shutterstock_1363648634-min.jpg>

**1. Continuous Development:**

This stage involves committing code to version control tools such as Git or SVN for maintaining the different versions of the code, and tools like Ant, Maven, Gradle for building/packaging the code into an executable file that can be forwarded to the QAs for testing.

**2. Continuous Integration:**

The stage is a critical point in the whole DevOps Lifecycle. It deals with integrating the different stages of the DevOps lifecycle and is, therefore, the key in automating the whole DevOps Process.

**3. Continuous Deployment:**

In this stage the code is built, the environment or the application is containerized and is pushed onto the desired server. The key processes in this stage are Configuration Management, Virtualization, and Containerization.

**4.** **Continuous Testing:**

The stage deals with automated testing of the application pushed by the developer. If there is an error, the message is sent back to the integration tool, this tool, in turn, notifies the developer of the error, If the test was a success, the message is sent to Integration-tool which pushes the build on the production server.

**5. Continuous Monitoring:**

The stage continuously monitors the deployed application for bugs or crashes. It can also be set up to collect user feedback. The collected data is then sent to the developers to improve the application.

### Why AWS for DevOps?

**Get Started Fast:**

Each AWS service is ready to use if you have an AWS account. There is no setup required or software to install.



Image 13: Get Started Fast

Reference: <https://aws.amazon.com/devops/>

**Fully Managed Services:**

These services can help you take advantage of AWS resources quicker. You can worry less about setting up, installing, and operating infrastructure on your own. This lets you focus on your core product.



Image 14: Fully Managed Services

Reference: <https://aws.amazon.com/devops/>

**Built for Scale:**

You can manage a single instance or scale to thousands using AWS services. These services help you make the most of flexible compute resources by simplifying provisioning, configuration, and scaling.



Image 15: Built for Scale

Reference: <https://aws.amazon.com/devops/>

**Programmable:**

You have the option to use each service via the AWS Command Line Interface or through APIs and SDKs. You can also model and provision AWS resources and your entire AWS infrastructure using declarative AWS CloudFormation templates.



Image 16: Programmable

Reference: <https://aws.amazon.com/devops/>

**Automation:**

AWS helps you use automation so you can build faster and more efficiently. Using AWS services, you can automate manual tasks or processes such as deployments, development & test workflows, container management, and configuration management.



Image 17: Automation

Reference: <https://aws.amazon.com/devops/>

**Secure:**

Use AWS Identity and Access Management (IAM) to set user permissions and policies. This gives you granular control over who can access your resources and how they access those resources.



Image 18: Secure

Reference: <https://aws.amazon.com/devops/>

**Large Partner Ecosystem:**

AWS supports a large ecosystem of partners which integrate with and extend AWS services. Use your preferred third-party and open-source tools with AWS to build an end-to-end solution. Visit here to learn more about our DevOps Partner Solutions.



Image 19: Large Partner Ecosystem

Reference: <https://aws.amazon.com/devops/>

**Pay-As-You-Go:**

With AWS purchase services as you need them and only for the period when you plan to use them. AWS pricing has no upfront fees, termination penalties, or long term contracts. The AWS Free Tier helps you get started with AWS.



Image 20: Pay-As-You-Go

Reference: <https://aws.amazon.com/devops/>

### Benefits of DevOps

**Speed:**

Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example, microservices and continuous delivery let teams take ownership of services and then release updates to them quicker.

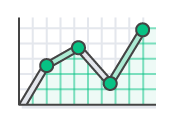


Image 21: Speed

Reference: <https://aws.amazon.com/devops/what-is-devops>

**Rapid Delivery:**

Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers’ needs and build competitive advantage. Continuous integration and continuous delivery are practices that automate the software release process, from build to deploy.

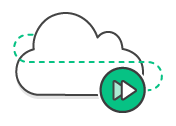


Image 22: Rapid Delivery

Reference: <https://aws.amazon.com/devops/what-is-devops>

**Reliability:**

Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like continuous integration and continuous delivery to test that each change is functional and safe. Monitoring and logging practices help you stay informed of performance in real-time.

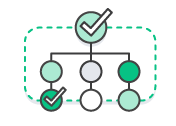


Image 23: Reliability

Reference: <https://aws.amazon.com/devops/what-is-devops>

**Scale:**

Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, infrastructure as code helps you manage your development, testing, and production environments in a repeatable and more efficient manner.

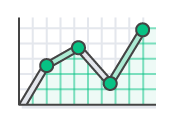


Image 24: Scale

Reference: <https://aws.amazon.com/devops/what-is-devops>

**Improved Collaboration:**

Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams collaborate closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).

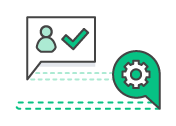


Image 25: Improved Collaboration

Reference: <https://aws.amazon.com/devops/what-is-devops>

**Security:**

Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and policy as code, you can define and then track compliance at scale.

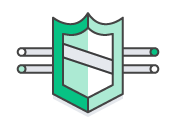


Image 26: Security

Reference: <https://aws.amazon.com/devops/what-is-devops>

### History of DevOps

Before DevOps, we had two approaches for software development namely the Waterfall and the Agile.

**Waterfall Model**

* The waterfall model is a software development model that is pretty straight forward and linear. This model follows a top-down approach.
* This model has various starting with Requirements gathering and analysis. This is the phase where you get the requirements from the client for developing an application. After this, you try to analyze these requirements.
* The next phase is the Design phase where you prepare a blueprint of the software. Here, you think about how the software is actually going to look like.
* Once the design is ready, you move further with the Implementation phase where you begin with the coding for the application. The team of developers works together on various components of the application.
* Once you complete the application development, you test it in the Verification phase. There are various tests conducted on the application such as unit testing, integration testing, performance testing, etc.
* After all the tests on the application are completed, it is deployed onto the production servers.
* At last, comes the Maintenance phase. In this phase, the application is monitored for performance. Any issues related to the performance of the application are resolved in this phase.

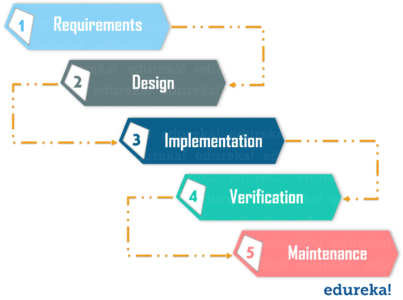


Image 27: Waterfall model

Reference: <https://www.edureka.co/blog/what-is-devops/>

**Advantages of the Waterfall Model:**

* Simple to understand and use
* Allows for easy testing and analysis
* Saves a significant amount of time and money
* Good for small projects if all requirements are clearly defined
* Allows for departmentalization & managerial control

**Disadvantages of Waterfall Model:**

* Risky and uncertain
* Lack of visibility of the current progress
* Not suitable when the requirements keep changing
* Difficult to make changes to the product when it is in the testing phase
* The end product is available only at the end of the cycle
* Not suitable for large and complex projects

**Agile Methodology**

Agile Methodology is an iterative based software development approach where the software project is broken down into various iterations or sprints. Each iteration has phases like the waterfall model such as Requirements Gathering, Design, Development, Testing, and Maintenance. The duration of each iteration is generally 2-8 weeks.

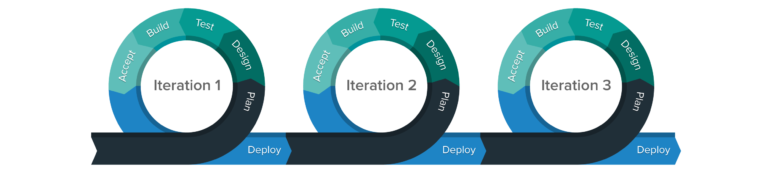


Image 28: Agile Methodology

Reference: <https://www.edureka.co/blog/what-is-devops/>

**Agile Process**

* In Agile, a company releases the application with some high priority features in the first iteration.
* After its release, the end-users or the customers give you feedback about the performance of the application.
* Then you make the necessary changes into the application along with some new features and the application is again released which is the second iteration.
* You repeat this entire procedure until you achieve the desired software quality.

**Advantages of Agile Model**

* It adaptively responds to requirement changes favorably
* Fixing errors early in the development process makes this process more cost-effective
* Improves the quality of the product and makes it highly error-free
* Allows for direct communication between people involved in software project
* Highly suitable for large & long-term projects
* Minimum resource requirements & very easy to manage

**Disadvantages of Agile Model**

* Highly dependent on clear customer requirements
* Quite Difficult to predict time and effort for larger projects
* Not suitable for complex projects
* Lacks documentation efficiency
* Increased maintainability risks

**DevOps vs Agile**

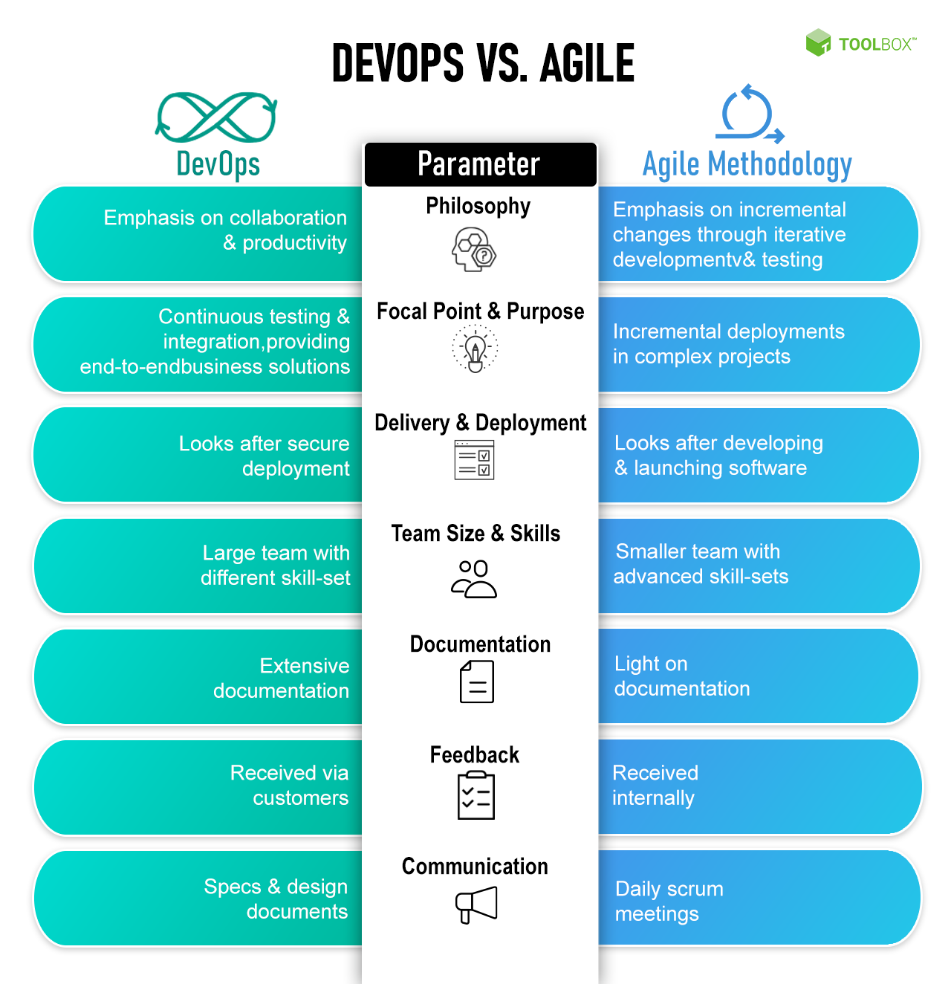


Image 29: DevOps vs Agile

Reference: <https://www.toolbox.com/tech/devops/articles/devops-vs-agile/>

### DevOps methodologies

DevOps is the direct descendant of agile software development, born from the need to keep up with increased software development velocity and throughput agile methods. Advancements in agile development highlighted the need for a more holistic approach to the software delivery life cycle, resulting in DevOps.

“Agile development” is an umbrella term for several iterative software development methodologies, many of which have carried over to DevOps:

* **Scrum**-a framework in which people can address complex adaptive problems while delivering products of the highest possible value.
* **Kanban**—a method for managing the creation of products with an emphasis on continual delivery while not overburdening the development team. Like Scrum, Kanban is a process designed to help teams work together more effectively.
* **Scaled Agile Framework (SAFe)**-a set of organization and workflow patterns intended to guide enterprises in scaling lean and agile practices. SAFe is one of a growing number of frameworks that seek to address the problems encountered when scaling beyond a single team.
* **Lean development**-a translation of lean manufacturing principles and practices to the software development domain. Lean offers a conceptual framework, values, and principles, as well as best practices derived from experience, that support agile organizations.
* **Extreme programming (XP)**-a software development methodology intended to improve software quality and responsiveness to changing customer requirements. XP advocates frequent releases in short development cycles, intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted. Other elements of extreme programming include programming in pairs or doing extensive code review, unit testing of all code, not programming of features until they are needed, a flat management structure, code simplicity and clarity, expecting changes in the customer’s requirements as time passes and the problem is better understood, and frequent communication with the customer.

### DevOps Tools

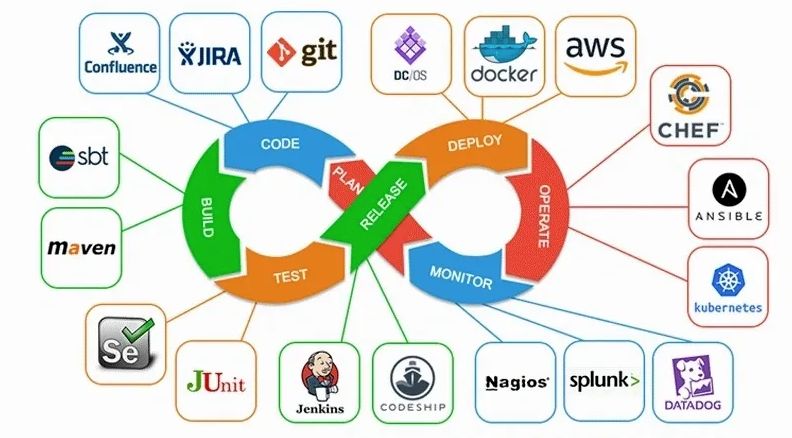


Image 30: DevOps Tools

Reference: <https://simpat.tech/wp-content/uploads/2021/09/devops-architecture-diagram.jpg>

DevOps tools lead to more collaborative teams, development pipelines that are faster, more robust applications and services, and more satisfied customers. We will be covering the following DevOps tools in this blog:

* Git
* Docker
* Selenium
* Jenkins
* Ansible
* Puppet
* Nagios
* Chef
* SVN
* Rational Clearcase
* Maven
* Apache Ant
* Kubernetes
* SignalFx
* Prometheus
* Splunk Cloud
* Raygun
* Catchpoint
* Splunk On-Call
* Gremlin
* JUnit

Quick and high-quality application delivery models have become the new standard of expectations. This triggered the rapid adoption of DevOps practices, which in turn, necessitated the high demand for DevOps tools. Here are some of the top DevOps tools:

**Git**

Git is an open-source, distributed, and the most popular software versioning system. Git has several sets of commands used to perform various operations. It works on a client-server model, which basically means that there is a central server that holds the main repository of code. Code can be downloaded from the main repository simultaneously by various clients or developers. Git is developed by Linus Torvalds. Git facilitates teams that are located at geographically different places to collaborate on the same project. Popular companies like Google, Facebook, Microsoft, Netflix extensively use Git in their CI/CD pipelines.

**Docker**

Docker is a containerization technology. Containers consist of all the applications with all of their dependencies. These containers can be deployed on any machine without caring about underlying host details. Containers can be a .net application or a website along with its dependencies like .net or lamp stack in case of a website application. These containers are used to automate the deployment process of applications in production and non-production environment.

**Selenium**

Selenium is an open-source, automated testing framework. Selenium is used majorly to automate the testing of web applications. We can define test in selenium which we want to test against our application. Selenium can repeatedly apply those tests to our application without manual intervention and generate reports. These reports can be used by testing individuals to see if the application is behaving correctly or not.

**Jenkins**

Jenkins is an open-source automation server written in java. Jenkins provides automation of the continuous delivery part. Jenkins is used in creating continuous delivery pipelines. To give you an example, here is a scenario wherein a Jenkins server will take our application container from the development environment and make it accessible to the testing environment, QA environment, or any other non-production environment in the beginning. Jenkins server is in the middle of the whole CI/CD pipeline. It automates this whole process which means whenever a developer commits a change in a code that code will automatically be visible to the testing server or QA team. They then can provide instantaneous feedback on these changes. Jenkins is used by Microsoft, Redhat, Rackspace, to name a few.

**Ansible**

Ansible is an open-source application that is used for automated software provisioning, configuration management, and application deployment. Ansible is the backbone of controlling an automated cluster environment consisting of many machines. Ansible works on the client-server model. The client acts as a master, which is the center point in our cluster and provides centralized control of all client machines (slaves) that are connected to it. We can give any command to any client machine or deploy any application to more than one machine from a single master machine. Ansible only requires SSH for communication so it does not need any software dependency to run. Ansible works on Unix.

**Puppet**

Puppet is an open-source software configuration management, automated provisioning tool. It is an alternative to Ansible and provides better control over client machines. Puppet comes up with GUI which makes it easier to use than Ansible. Puppet is cross-platform, it runs on both Unix and Microsoft Windows. Puppet uses a manifest file and applies those specifications across all machines. Unlike Ansible, Puppet is an agent-based tool. The Puppet master runs on a master machine and the Puppet agent runs on all client machines. Puppet is used by Microsoft, Google, Accenture, etc.

**Nagios**

Nagios is used for continuous monitoring of infrastructure. Nagios provides server monitoring, application monitoring, and network monitoring. With Nagios, we can monitor a whole data center from a single server. We can see whether switches are working correctly, servers are not having too much load, or if any part of the application is down. It provides a nice GUI interface to check various details like how much memory is used, what is the fan speed, routing tables of switches, or the state of the SQL server. Nagios has a modular design. It supports NRPE plugins which can be used to add monitoring parameters on existing Nagios. There are various plugins available on the internet which can be used freely to add features to Nagios. Nagios is the most popular tool in continuous monitoring.

**Chef**

Chef is a configuration management tool. Chef is used to manage configurations like creating or removing a user, adding an SSH key to a user present on multiple nodes, installing or removing a service, etc. We can manage up to 10,000 nodes by using chef. These changes are pushed by cookbooks or recipes. Chef has three components viz. Chef server, workstation, and nodes. The Chef server is a central point where all details of our Chef infrastructure reside. Chef workstation holds recipes or cookbooks which pushes particular configurations to our chef infrastructure. Nodes are simple machines that are configured using chef. Chef has API support from AWS, Azure, Rackspace, which makes it easy to use with infrastructure-as-code methodology.

**SVN**

SVN or Subversion is an open-source, centralized software versioning & revision control system. SVN is an alternative to GIT. SVN is a centralized system which means every time if a team member or client wants to make changes in the code, he has to inform the central server or repo about these changes. To use an SVN repo, the client needs to connect to the central server. It can check out the code to download the repo from the server. The client then moves on to make changes in this code and commit this change by informing the central repo. From the central repo, these changes will be visible to other team members or clients.

**Rational ClearCase**

Rational ClearCase manages changes across the software lifecycle. ClearCase is used in Software configuration management of source code. ClearCase is used in both hardware and software development. The center of ClearCase is a secure data repository. It has data that is shared by all users including accounting data and historical data on development processes themselves. It shows details like which user implemented which version, when, and why. There are 3 products of ClearCase—Rational ClearCase is for medium to large teams, Rational ClearCase LT is useful for small to medium teams, and Rational ClearCase multisite is used by geographically distributed teams.

**Maven**

Maven is a build automation tool. It automates the software build process & dependencies resolution. A Maven project is configured using a project object model or POM.XML file which describes the build process and the software project, its dependencies on external modules and components, build order, directories. Maven can dynamically download these external modules & Maven plugins during the build process itself. Maven can build and manage projects on Java, C#, Scala, Ruby, and other languages. Maven is majorly used by Apache Foundation to automate the build of some of its large projects like Apache Hadoop.

**Apache Ant**

Apache Ant is a software tool used for the automation of the software build process. Apache Ant is inspired by Unix make utility. Apache Ant uses an XML file build.xml in place of makefile which is used by make utility for build processes. It automates repetitive tasks in the build process and generates documentation. Ant builds are based on three blocks viz. tasks, targets, and extension points. Ant supports many third-party extensions like Eclipse IDE and NetBeans IDE.

**Kubernetes**

Kubernetes is an open-source container orchestration tool. It is developed by Google. It is used in continuous deployment and auto-scaling of container clusters. It increases fault tolerance, load balancing in a container cluster. Kubernetes maintains the desired state of a cluster; this desired state is described in the YAML file. YAML file contains the state of pods or slave nodes and replication unit for a cluster. Kubernetes uses this YAML file to maintain the desired state of the cluster. For example, in case, a pod is serving more requests than another pod, then it can automatically distribute the load to other pods. In case one machine fails, then it can configure another pod to replace it thus ensuring fault tolerance, load balancing, and high availability in a cluster. Kubernetes is used in high-performance data centers like that of Google, Facebook, and Amazon Web Services.

**SignalFx**

SignalFx, which recently got acquired by Splunk, is a complete observability tool. It can collect traces, metrics, and events from applications and infrastructure to help inform users of not only the system’s health but why it is behaving a certain way. This helps teams to fix issues faster as well as connect application and infrastructure monitoring with the requirements of the business. SignalFx is great to use for debugging and post-incident reviews through high cardinality analytics, service mapping, and detailed visualizations and dashboards.

**Prometheus**

Prometheus is an open-source, time-series database and monitoring tool that is mostly used by DevOps and IT teams. It generates alerts based on time-series data. One can generate precise alerts and visualizations to get business insights and engineering outcomes. Developers and IT practitioners can easily customize the tool for their own use cases. Following are some of its features:

* Functional sharding and federation helps with scaling
* Easy service instrumentation with numerous client libraries
* PromQL enables powerful reporting capabilities

**Splunk Cloud**

Splunk Enterprise and Splunk Cloud are log management, infrastructure monitoring, and application monitoring tools. It can collect data from services and devices as well as other monitoring tools such as DevOps monitoring tools. Splunk serves as the single source of truth for the health and performance of a system. Its powerful log search, filtration functionality, and informative visualizations and dashboards lead to quicker incident resolution. Splunk provides observability and provides the tools required to take action whether there is an on-premises architecture or in a cloud environment. Splunk when paired with VictorOps creates complete observability, data-driven incident management, and on-call incident response.

**Raygun**

Raygun generates real insights into the way users experience a service. It has the ability to provide detailed reports on everything like full-scale app crashes, downtime, performance metrics like load speeds, network latency, etc. Its real-user monitoring can identify and expose both client-side and server-side problems for users as well as help product teams give priority to engineering roadmaps to align with real problems. The APM tool works well with Raygun’s error management workflow. It automatically links errors back to the source code. This brings Development and Operations together through a single source of truth for the entire team—performance problems and the cause of errors.

**Catchpoint**

The Catchpoint monitoring tool combines synthetic monitoring, real-user monitoring, network monitoring, and endpoint monitoring for the detection of errors and incidents anywhere in an architecture. The ability to run synthetic metrics through the system is a unique advantage as compared to others. For smaller teams who don’t have large amounts of real metrics, it can help you find service reliability or performance problems as you scale before customers notice it. Catchpoint helps identify issues whether it’s caused by a user’s browser or device or an application or infrastructure problem.

**Splunk On-Call**

With Splunk On-Call, one can alert engineers and on-call responders about problems and incidents in real-time as well as provide contextual alert information and remediation instructions. In one single pane of glass, DevOps and IT practitioners can collaborate to drastically reduce the time taken to acknowledge and resolve incidents. With reports like MTTA/MTTR, Post-Incident Review, Incident Frequency, teams can drive swift problem resolution, manage alert noise, reduce burnout, etc.

**Gremlin**

Netflix popularized chaos engineering as a means to simulate chaos through systems in order to test the response to stress and unpredictable events. Gremlin lets you design experiments according to your liking to conduct or simply re-enact issues experienced in the past and run them through your applications and services to see how they handle it.

**JUnit**

JUnit is the de-facto test automation tool for Java. No matter if JUnit is used for writing unit or integration tests or even if another framework is used in conjunction with it - such as Selenium - the output results file can be uploaded to Zephyr Scale for generating reports and leveraging all the other capabilities

**What is the best DevOps tool?**

The best DevOps tools are the ones that serve the processes and people that form your DevOps culture. DevOps is not a product you can buy and implement and say, “Now we’ve got DevOps.” DevOps technologist Alex Honor published a post in the early days of DevOps titled “People Over Process Over Tools,” crystalizing the movement’s emphasis on culture over tooling.

That said, there are a number of notable products that help DevOps teams get the job done. A non-exhaustive list of prominent tools, both open-source and proprietary, includes:

* **Source code management:** Git (GitLab, GitHub), Bitbucket
* **Configuration management:** Puppet, Chef, Ansible, CFEngine
* **Release management:** Jenkins, Travis, CircleCl, TeamCity, Gradle, Bamboo
* **Orchestration:** Mesos, Zookeeper, Kubernetes
* **Monitoring, virtualization, containerization:** Nagios, Icignia, Monit, OpenStack, Vagrant, AWS, Docker, Kubernetes
* **Log and application lifecycle analytics:** Splunk is a leading log management tool ideal for DevOps.

**DevOps Tooling by AWS**

AWS provides services that help you practice DevOps at your company and that are built first for use with AWS. These tools automate manual tasks, help teams manage complex environments at scale, and keep engineers in control of the high velocity that is enabled by DevOps.

* Continuous Integration and Continuous Delivery
* Microservices
* Infrastructure as Code
* Monitoring and Logging
* Platform as a Service
* Version Control

**Continuous Integration and Continuous Delivery**

The AWS Developer Tools help you securely store and version your application's source code and automatically build, test, and deploy your application to AWS or your on-premises environment.

Start with AWS CodePipeline to build a continuous integration or continuous delivery workflow that uses AWS CodeBuild, AWS CodeDeploy, and other tools, or use each service separately.

**Software Release Workflows**

**AWS CodePipeline-**AWS CodePipeline is a continuous integration and continuous delivery service for fast and reliable application and infrastructure updates. CodePipeline builds, tests, and deploys your code every time there is a code change, based on the release process models you define. This enables you to rapidly and reliably deliver features and updates.

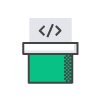


Image 31: Software Release Workflows

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Build and Test Code**

**AWS CodeBuild**-AWS CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don’t need to provision, manage, and scale your own build servers. CodeBuild scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue.

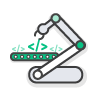


Image 32: Build and Test Code

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Deployment Automation**

**AWS CodeDeploy-**AWS CodeDeploy automates code deployments to any instance, including Amazon EC2 instances and on-premises servers. AWS CodeDeploy makes it easier for you to rapidly release new features, helps you avoid downtime during application deployment, and handles the complexity of updating your applications.



Image 33: Deployment Automation

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Unified CI/CD Projects**

**AWS CodeStar-**AWS CodeStar enables you to quickly develop, build, and deploy applications on AWS. AWS CodeStar provides a unified user interface, enabling you to easily manage your software development activities in one place. With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster.



Image 34: Unified CI/CD Projects

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Microservices**

Build and deploy a microservices architecture using containers or serverless computing.

**Production Docker Platform**

**Amazon Elastic Container Service-**Amazon Elastic Container Service (ECS) is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.



Image 35: Production Docker Platform

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Serverless Computing**

**AWS Lambda**-AWS Lambda lets you run code without provisioning or managing servers. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability.



Image 36: Serverless Computing

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Infrastructure as Code**

Provision, configure, and manage your AWS infrastructure resources using code and templates. Monitor and enforce infrastructure compliance.

**Templated Infrastructure Provisioning**

**AWS CloudFormation**-AWS CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, provisioning and updating them in an orderly and predictable fashion. You can use AWS CloudFormation’s sample templates or create your own templates.

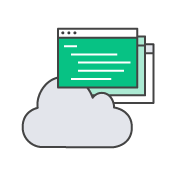


Image 37: Templated infrastructure Provisioning

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Chef Configuration Management**

**AWS OpsWorks**-AWS OpsWorks is a configuration management service that uses Chef, an automation platform that treats server configurations as code. OpsWorks uses Chef to automate how servers are configured, deployed, and managed across your Amazon Elastic Compute Cloud (Amazon EC2) instances or on-premises compute environments. OpsWorks has two offerings, AWS Opsworks for Chef Automate, and AWS OpsWorks Stacks.

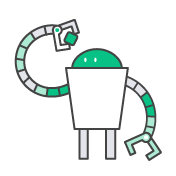


Image 38: Chef Configuration Management

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Configuration Management**

**AWS Systems Manager**-AWS Systems Manager is a management service that helps you automatically collect software inventory, apply OS patches, create system images, and configure Windows and Linux operating systems. These capabilities help you define and track system configurations, prevent drift, and maintain software compliance of your EC2 and on-premises configurations.

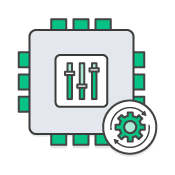


Image 39: Configuration Management

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Policy as Code**

**AWS Config**-AWS Config is a fully managed service that provides you with an AWS resource inventory, configuration history, and configuration change notifications to enable security and governance. Config Rules enables you to create rules that automatically check the configuration of AWS resources recorded by AWS Config.



Image 40: Policy as Code

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Monitoring and Logging**

Record logs and monitor application and infrastructure performance in near real-time.

**Cloud and Network Monitoring**

**Amazon CloudWatch**-Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to collect and track metrics, collect and monitor log files, set alarms, and automatically react to changes in your AWS resources.



Image 41: Cloud and Network Monitoring

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Distributed Tracing**

**AWS X-Ray**-AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. With X-Ray, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors.



Image 42: Distributed Racing

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Activity & API Usage Tracking**

**AWS CloudTrail-**AWS CloudTrail is a web service that records AWS API calls for your account and delivers log files to you. The recorded information includes the identity of the API caller, the time of the API call, the source IP address of the API caller, the request parameters, and the response elements returned by the AWS service.



Image 43: Activity & API Usage Tracking

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Platform as a Service**

Deploy web applications without needing to provision and manage the infrastructure and application stack.

**Run and Manage Web Apps**

**AWS Elastic Beanstalk**-AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

You can simply upload your code and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring. At the same time, you retain full control over the AWS resources powering your application and can access the underlying resources at any time.



Image 44: Run and Manage Web Apps

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**Version Control**

Host secure, highly scalable Git repositories in the cloud.

**Private Git Hosting**

**AWS CodeCommit-**AWS CodeCommit is a fully-managed source control service that makes it easy for companies to host secure and highly scalable private Git repositories. You can use CodeCommit to securely store anything from source code to binaries, and it works seamlessly with your existing Git tools.

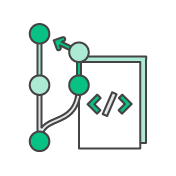


Image 45: Private Git Hosting

Reference: [https://aws.amazon.com/devops/](https://aws.amazon.com/devops/what-is-devops)

**IBMCloud DevOps tools: Building a DevOps toolchain**

The demands of DevOps and DevOps culture put a premium on tooling that supports asynchronous collaboration, seamlessly integrates DevOps workflows, and automates the entire DevOps lifecycle as much as possible. Categories of DevOps tools include:

**Project management tools:**

Tools that enable teams to build a backlog of user stories (requirements) that form coding projects, break them down into smaller tasks and track the tasks through to completion. Many supports agile project management practices such, as Scrum, Lean and Kanban, that developers bring to DevOps. Popular open-source options include GitHub Issues and Jira.

**Collaborative source code repositories:**

Version-controlled coding environments that that let multiple developers to work on the same code base. Code repositories should integrate with CI/CD, testing and security tools, so that when code is committed to the repository it can automatically move to the next step. Open-source code repositories include GiHub and GitLab.

**CI/CD pipelines:**

Tools that automate code checkout, building, testing and deployment. Jenkins is the most popular open-source tool in this category; many previously open-source alternatives, such as CircleCI, are now available in commercial versions only. When it comes to continuous deployment (CD) tools, Spinnaker straddles between application and infrastructure as code layers. ArgoCD is another popular open-source choice for Kubernetes native CI/CD.

**Test automation frameworks:**

These include software tools, libraries and best practices for automating unit, contract, functional, performance, usability, penetration and security tests. The best of these tools support multiple languages; some use artificial intelligence (AI) to automatically reconfigure tests in response to code changes. The expanse of test tools and frameworks is far and wide! Popular open-source test automation frameworks include Selenium, Appium, Katalon, Robot Framework, and Serenity (formerly known as Thucydides).

**Configuration management (infrastructure as code) tools:**

These enable DevOps engineers to configure and provision fully versioned and fully documented infrastructure by executing a script. Open-source options include Ansible (Red Hat), Chef, Puppet and Terraform. Kubernetes performs the same function for containerized applications (see 'DevOps and cloud-native development,' below).

**Monitoring tools:**

These help DevOps teams identify and resolve system issues; they also gather and analyze data in real time to reveal how code changes impact application performance. Open-source monitoring tools include Datadog, Nagios, Prometheus and Splunk.

**Continuous feedback tools:**

Tools that gather feedback from users, either through heatmapping (recording users' actions on screen), surveys, or self-service issue ticketing.

### DevOps Services

**AWS**

* AWS CodePipeline
* AWS CodeBuild
* AWS CodeDeploy
* AWS CodeStar

**AWS CodePipeline**

AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. CodePipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. This enables you to rapidly and reliably deliver features and updates. You can easily integrate AWS CodePipeline with third-party services such as GitHub or with your own custom plugin. With AWS CodePipeline, you only pay for what you use. There are no upfront fees or long-term commitments

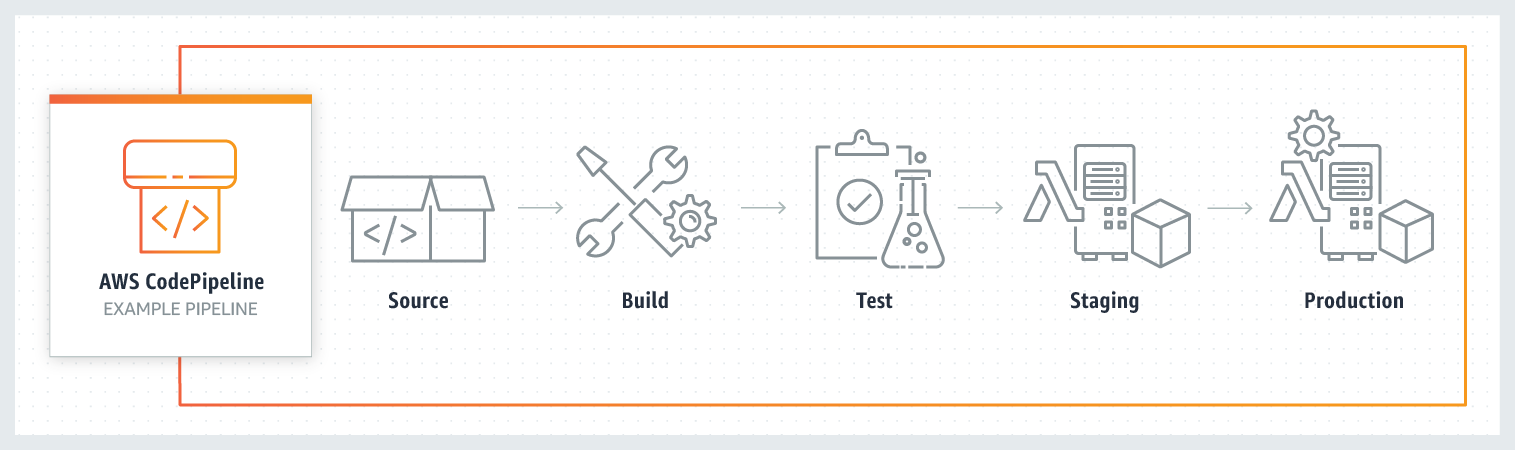


Image 46: Code pipeline

Reference: <https://aws.amazon.com/codepipeline/>

**Benefits**

**Rapid delivery**

AWS CodePipeline automates your software release process, allowing you to rapidly release new features to your users. With CodePipeline, you can quickly iterate on feedback and get new features to your users faster.

Automating your build, test, and release process allows you to quickly and easily test each code change and catch bugs while they are small and simple to fix. You can ensure the quality of your application or infrastructure code by running each change through your staging and release process.

**Configurable workflow**

AWS CodePipeline allows you to model the different stages of your software release process using the console interface, the AWS CLI, AWS CloudFormation, or the AWS SDKs. You can easily specify the tests to run and customize the steps to deploy your application and its dependencies.

**Get started fast**

With AWS CodePipeline, you can immediately begin to model your software release process. There are no servers to provision or set up. CodePipeline is a fully managed continuous delivery service that connects to your existing tools and systems.

**Easy to integrate**

AWS CodePipeline can easily be extended to adapt to your specific needs. You can use our pre-built plugins or your own custom plugins in any step of your release process. For example, you can pull your source code from GitHub, use your on-premises Jenkins build server, run load tests using a third-party service, or pass on deployment information to your custom operations dashboard.

**AWS CodeBuild**

AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don’t need to provision, manage, and scale your own build servers. CodeBuild scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue. You can get started quickly by using prepackaged build environments, or you can create custom build environments that use your own build tools. With CodeBuild, you are charged by the minute for the compute resources you use.

**Benefits**

Fully managed build service

AWS CodeBuild eliminates the need to set up, patch, update, and manage your own build servers and software. There is no software to install or manage.

**Continuous scaling**

AWS CodeBuild scales up and down automatically to meet your build volume. It immediately processes each build you submit and can run separate builds concurrently, which means your builds are not left waiting in a queue.

**Pay as you go**

With AWS CodeBuild, you are charged based on the number of minutes it takes to complete your build. This means you no longer have to worry about paying for idle build server capacity.

**Extensible**

You can bring your own build tools and programming runtimes to use with AWS CodeBuild by creating customized build environments in addition to the prepackaged build tools and runtimes supported by CodeBuild.

**Enables continuous integration and delivery**

AWS CodeBuild belongs to a family of AWS Code Services, which you can use to create complete, automated software release workflows for continuous integration and delivery (CI/CD). You can also integrate CodeBuild into your existing CI/CD workflow. For example, you can use CodeBuild as a worker node for your existing Jenkins server setup for distributed builds.

**Secure**

With AWS CodeBuild, your build artifacts are encrypted with customer-specific keys that are managed by the AWS Key Management Service (KMS). CodeBuild is integrated with AWS Identity and Access Management (IAM), so you can assign user-specific permissions to your build projects

**AWS CodeDeploy**

AWS CodeDeploy is a fully managed deployment service that automates software deployments to a variety of compute services such as Amazon EC2, AWS Fargate, AWS Lambda, and your on-premises servers. AWS CodeDeploy makes it easier for you to rapidly release new features, helps you avoid downtime during application deployment, and handles the complexity of updating your applications. You can use AWS CodeDeploy to automate software deployments, eliminating the need for error-prone manual operations. The service scales to match your deployment needs.

**Benefits**

**Automated deployments**

AWS CodeDeploy fully automates your software deployments, allowing you to deploy reliably and rapidly. You can consistently deploy your application across your development, test, and production environments whether deploying to Amazon EC2, AWS Fargate, AWS Lambda, or your on-premises servers. The service scales with your infrastructure.

**Minimize downtime**

AWS CodeDeploy helps maximize your application availability during the software deployment process. It introduces changes incrementally and tracks application health according to configurable rules. Software deployments can easily be stopped and rolled back if there are errors.

**Centralized control**

AWS CodeDeploy allows you to easily launch and track the status of your application deployments through the AWS Management Console or the AWS CLI. CodeDeploy gives you a detailed report allowing you to view when and to where each application revision was deployed. You can also create push notifications to receive live updates about your deployments.

**Easy to adopt**

AWS CodeDeploy is platform and language agnostic, works with any application, and provides the same experience whether you’re deploying to Amazon EC2, AWS Fargate, or AWS Lambda. You can easily reuse your existing setup code. CodeDeploy can also integrate with your existing software release process or continuous delivery toolchain (e.g., AWS CodePipeline, GitHub, Jenkins).

**AWS CodeStar**

AWS CodeStar enables you to quickly develop, build, and deploy applications on AWS. AWS CodeStar provides a unified user interface, enabling you to easily manage your software development activities in one place. With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster. AWS CodeStar makes it easy for your whole team to work together securely, allowing you to easily manage access and add owners, contributors, and viewers to your projects. Each AWS CodeStar project comes with a project management dashboard, including an integrated issue tracking capability powered by Atlassian JIRA Software. With the AWS CodeStar project dashboard, you can easily track progress across your entire software development process, from your backlog of work items to teams’ recent code deployments. Visit here to learn more.

There is no additional charge for using AWS CodeStar. You only pay for the AWS resources that you provision for developing and running your application (for example, Amazon EC2 instances).

**Benefits**

**Start developing on AWS in minutes**

AWS CodeStar makes it easy for you to set up your entire development and continuous delivery toolchain for coding, building, testing, and deploying your application code. To start a project, you can choose from a variety of AWS CodeStar templates for Amazon EC2, AWS Lambda, and AWS Elastic Beanstalk. You have the option to choose AWS CodeCommit or GitHub to use as your project’s source control. You also have the option to edit your source code using one of several options including AWS Cloud9, Microsoft Visual Studio, or Eclipse. After you make your selections the underlying AWS services are provisioned in minutes, allowing you to quickly start coding and deploying your applications.

**Manage software delivery in one place**

AWS CodeStar provides an easy way to coordinate your day-to-day development activities through a unified user interface, reducing the need to switch between various service consoles. AWS CodeStar’s project dashboard lets you monitor application activity, and track progress across all stages of your software development process, including code commits, builds, tests, and deployments, from a central place. AWS CodeStar integrates Atlassian JIRA Software, a third-party issue tracking and project management tool, allowing you to easily manage JIRA issues directly in the AWS CodeStar dashboard.

**Work across your team securely**

AWS CodeStar enables you to collaborate on projects across your team in a secure manner. You can easily manage access for project owners, contributors, and viewers without needing to manually configure your own policy for each service. AWS CodeStar simplifies the process of setting up project access for teams by providing built-in role-based policies that follow AWS Identity and Access Management best practices.

**Choose from a variety of project templates**

With AWS CodeStar project templates, you can easily develop a variety of applications such as websites, web applications, web services, and Alexa skills. AWS CodeStar project templates include the code for getting started on supported programming languages including Java, JavaScript, PHP, Ruby, C#, and Python.

**DevOps services on IBM Cloud**

**IBM Cloud Continuous Delivery**

Embrace enterprise-ready DevOps. Create toolchains, automate builds, tests, deployments and more.

**IBM Cloud App Configuration**

Centralize feature management and configuration.

**IBM Cloud Schematics**

Configure and automate management of IBM Cloud resources.

**IBM Key Protect**

Get visibility and control of entire key lifecycle.

**IBM Cloud Secrets Manager**

Centrally manage secrets used in your apps and services.

**IBM Kubernetes Service**

Deploy and operate a Kubernetes cluster on IBM Cloud

**IBM Cloud Satellite™**

Run IBM Cloud services on your infrastructure with consistency.

**IBM Cloud Code Engine**

Run your application, batch jobs or container on a managed serverless platform.

**IBM Cloud Monitoring**

Get in-depth visibility into infrastructure and app performance.

**IBM Cloud Security and Compliance Center**

Govern cloud resource configurations and centrally manage your compliance with organization and regulatory guidelines.

**IBM Cloud Activity Tracker**

Manage compliance controls within the IBM Cloud platform.

### Disadvantages of DevOps

* Demands proper mindset across the company
* Lowered business security by outsourcing the DevOps operations
* Dealing with the legacy system is a challenge
* Practicing security for CI/CD is a separate affair
* Getting the right pool of DevOps expertise is a challenge
* Challenges with the number of tools and switching tools
* Transition challenges (organizational and technical)

# AWS Code Commit, Deploy and Pipeline

## AWS CodeCommit

AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. It makes it easy for teams to securely collaborate on code with contributions encrypted in transit and at rest. CodeCommit eliminates the need for you to manage your own source control system or worry about scaling its infrastructure. You can use CodeCommit to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with your existing Git-based tools.

**Benefits**

**Fully managed**

AWS CodeCommit eliminates the need to host, maintain, back up, and scale your own source control servers. The service automatically scales to meet the growing needs of your project.

**Secure**

AWS CodeCommit automatically encrypts your files in transit and at rest. CodeCommit is integrated with AWS Identity and Access Management (IAM) allowing you to customize user-specific access to your repositories.

**High availability**

AWS CodeCommit has a highly scalable, redundant, and durable architecture. The service is designed to keep your repositories highly available and accessible.

**Collaborate on code**

AWS CodeCommit helps you collaborate on code with teammates via pull requests, branching, and merging. You can implement workflows that include code reviews and feedback by default, and control who can make changes to specific branches.

**Faster development lifecycle**

AWS CodeCommit keeps your repositories close to your build, staging, and production environments in the AWS cloud. You can transfer incremental changes instead of the entire application. This allows you to increase the speed and frequency of your development lifecycle.

**Use your existing tools**

AWS CodeCommit supports all Git commands and works with your existing Git tools. You can keep using your preferred development environment plugins, continuous integration/continuous delivery systems, and graphical clients with CodeCommit.

AWS CodeCommit is a secure online version control service which hosts private Git repositories. A team need not maintain their own version control repository instead they use AWS CodeCommit to store their source code or even binaries like the WAR/JAR/EAR files generated out of the build.

With AWS CodeCommit you create a repository and every developer will clone it to their local machine, add files to it and push it back to the AWS CodeCommit repository. One uses the standard GIT commands with the AWS CodeCommit repository.

For E.g. once the AWS CodeCommit repository is cloned to local machine you would use commands like ‘git pull’, ‘git add’, ‘git commit’, ‘git push’ etc..

Illustrative AWS CodeCommit empty repository created

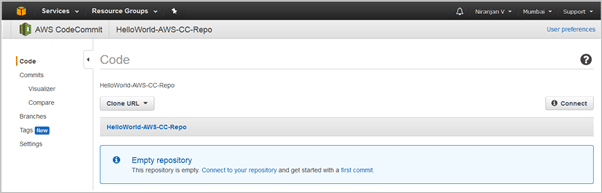
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image020.png)

Image 47: Code commit Empty Repository

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Clone the repository to the local machine

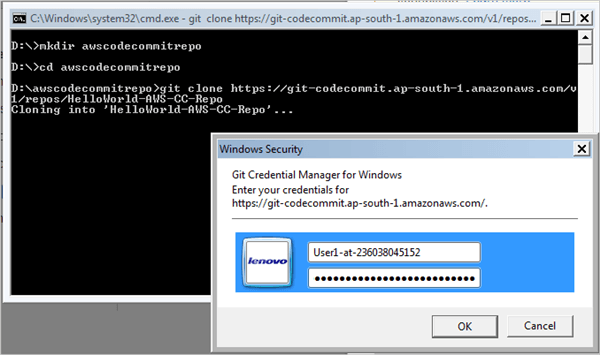
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image021.png)

Image 48: Clone Repository

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Files added to AWS CodeCommit repository

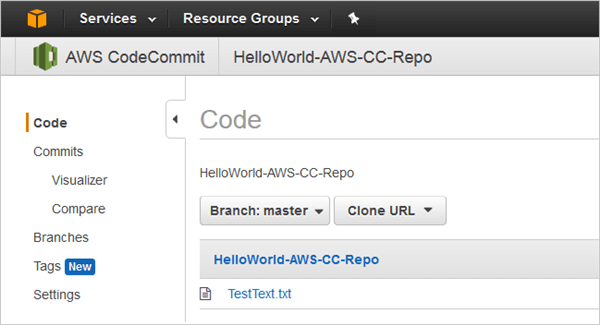
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image022.png)

Image 49: Code Commit Files Add

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

## AWS CodeBuild

AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don’t need to provision, manage, and scale your own build servers. CodeBuild scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue. You can get started quickly by using prepackaged build environments, or you can create custom build environments that use your own build tools. With CodeBuild, you are charged by the minute for the compute resources you use.

**Benefits**

Fully managed build service

AWS CodeBuild eliminates the need to set up, patch, update, and manage your own build servers and software. There is no software to install or manage.

**Continuous scaling**

AWS CodeBuild scales up and down automatically to meet your build volume. It immediately processes each build you submit and can run separate builds concurrently, which means your builds are not left waiting in a queue.

**Pay as you go**

With AWS CodeBuild, you are charged based on the number of minutes it takes to complete your build. This means you no longer have to worry about paying for idle build server capacity.

**Extensible**

You can bring your own build tools and programming runtimes to use with AWS CodeBuild by creating customized build environments in addition to the prepackaged build tools and runtimes supported by CodeBuild.

**Enables continuous integration and delivery**

AWS CodeBuild belongs to a family of AWS Code Services, which you can use to create complete, automated software release workflows for continuous integration and delivery (CI/CD). You can also integrate CodeBuild into your existing CI/CD workflow. For example, you can use CodeBuild as a worker node for your existing Jenkins server setup for distributed builds.

**Secure**

With AWS CodeBuild, your build artifacts are encrypted with customer-specific keys that are managed by the AWS Key Management Service (KMS). CodeBuild is integrated with AWS Identity and Access Management (IAM), so you can assign user-specific permissions to your build projects

To implement Continuous Integration AWS CodeBuild like Jenkins fetches the latest changes of the source code from AWS CodeCommit or GitHub repository as configured and based on the build specification YAML file (created as buildspec.yml) the commands are run based on the four phases like Install, Pre-build, Build and Post-build.

Once the build is completed the artifacts (WAR/ZIP/JAR/EAR) are stored in the AWS Storage which is an S3 bucket.

Samplebuildspec.yml file

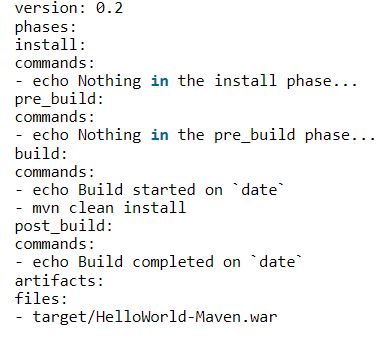


Image 50: Yaml file

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Sample AWS Codebuild project

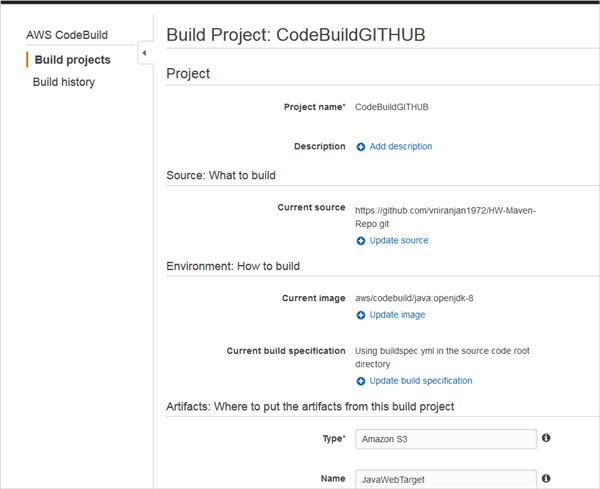
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image023.png)

Image 51: Code Build Sample

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Build Success

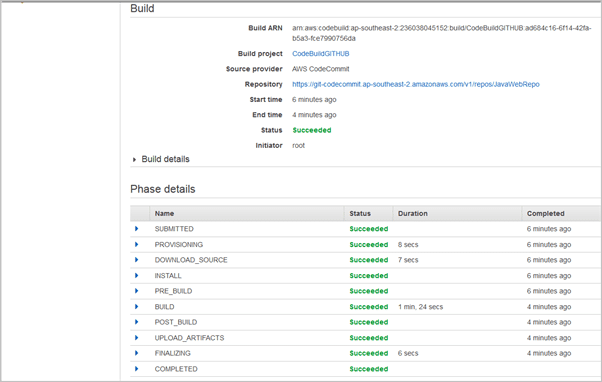
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image024.png)

Image 52: Code Build Success

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Artifact (WAR file) copied to S3 bucket

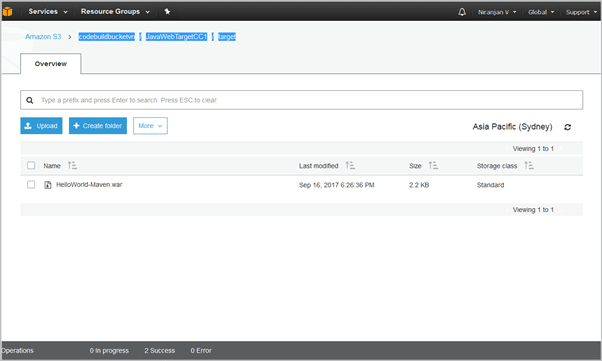
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image025-1.png)

Image 53: WAR File copied to s3 bucket

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

## AWS CodeDeploy

AWS CodeDeploy is a fully managed deployment service that automates software deployments to a variety of compute services such as Amazon EC2, AWS Fargate, AWS Lambda, and your on-premises servers. AWS CodeDeploy makes it easier for you to rapidly release new features, helps you avoid downtime during application deployment, and handles the complexity of updating your applications. You can use AWS CodeDeploy to automate software deployments, eliminating the need for error-prone manual operations. The service scales to match your deployment needs.

**Benefits**

**Automated deployments**

AWS CodeDeploy fully automates your software deployments, allowing you to deploy reliably and rapidly. You can consistently deploy your application across your development, test, and production environments whether deploying to Amazon EC2, AWS Fargate, AWS Lambda, or your on-premises servers. The service scales with your infrastructure.

**Minimize downtime**

AWS CodeDeploy helps maximize your application availability during the software deployment process. It introduces changes incrementally and tracks application health according to configurable rules. Software deployments can easily be stopped and rolled back if there are errors.

**Centralized control**

AWS CodeDeploy allows you to easily launch and track the status of your application deployments through the AWS Management Console or the AWS CLI. CodeDeploy gives you a detailed report allowing you to view when and to where each application revision was deployed. You can also create push notifications to receive live updates about your deployments.

**Easy to adopt**

AWS CodeDeploy is platform and language agnostic, works with any application, and provides the same experience whether you’re deploying to Amazon EC2, AWS Fargate, or AWS Lambda. You can easily reuse your existing setup code. CodeDeploy can also integrate with your existing software release process or continuous delivery toolchain (e.g., AWS CodePipeline, GitHub, Jenkins).

As the name suggests AWS Codedeploy is the deployment service which automates the deployment of the application (in this case WAR file) to the Amazon EC2 Linux or Windows instances.

Since we now have the artifacts stored in S3 bucket which was completed using AWS CodeBuild the artifacts are then picked up from the S3 bucket and deployed appropriately to the app server Tomcat or JBoss etc. in the AWS EC2 instance provisioning.

AWS CodeDeploy depends on a YAML file called appspec.yml which has instructions on the deployment to the EC2 instances.

Sample appspec.yml file where the index.html file is copied and deployed to the Apache server

**before\_install** script

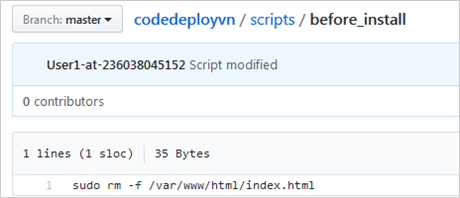
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image026.png)

Image 54: Before Install script

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

**restart\_server** script

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image027.png)

Image 55: Restart Server script

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

GitHub repo of all files needed to run AWS CodeDeploy

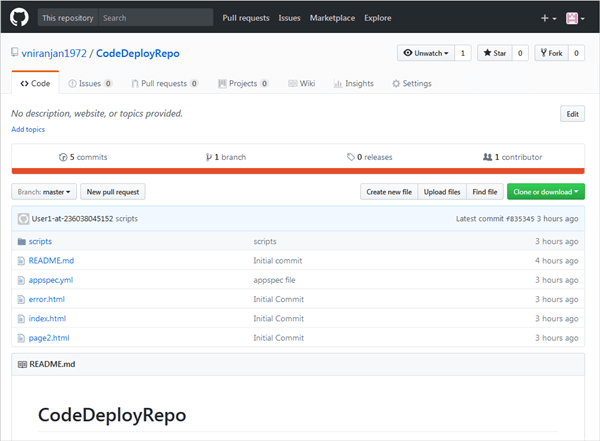
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image028.png)

Image 56: GitHub Repository

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Deployment execution in AWS CodeDeploy

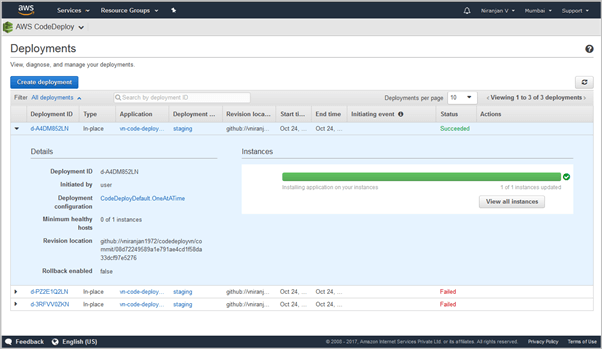
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image029.png)

Image 57: Code Deploy-Deployment Execution

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

## AWS CodePipeline

AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates. CodePipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define. This enables you to rapidly and reliably deliver features and updates. You can easily integrate AWS CodePipeline with third-party services such as GitHub or with your own custom plugin. With AWS CodePipeline, you only pay for what you use. There are no upfront fees or long-term commitments

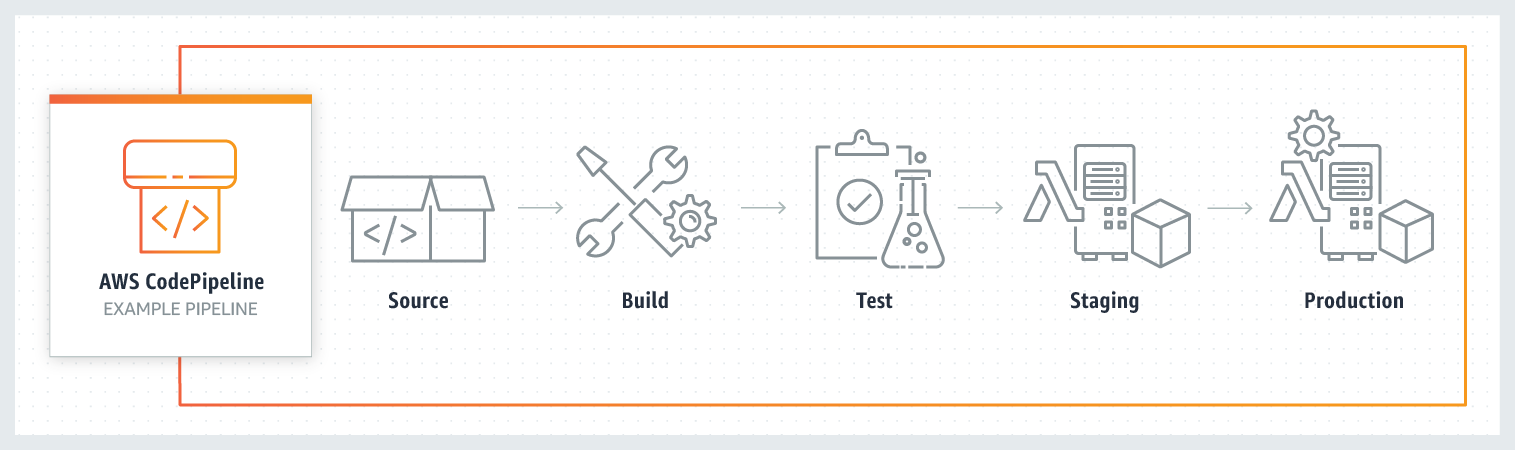


Image 58: Code pipeline

Reference: <https://aws.amazon.com/codepipeline/>

**Benefits**

**Rapid delivery**

AWS CodePipeline automates your software release process, allowing you to rapidly release new features to your users. With CodePipeline, you can quickly iterate on feedback and get new features to your users faster.

Automating your build, test, and release process allows you to quickly and easily test each code change and catch bugs while they are small and simple to fix. You can ensure the quality of your application or infrastructure code by running each change through your staging and release process.

**Configurable workflow**

AWS CodePipeline allows you to model the different stages of your software release process using the console interface, the AWS CLI, AWS CloudFormation, or the AWS SDKs. You can easily specify the tests to run and customize the steps to deploy your application and its dependencies.

**Get started fast**

With AWS CodePipeline, you can immediately begin to model your software release process. There are no servers to provision or set up. CodePipeline is a fully managed continuous delivery service that connects to your existing tools and systems.

**Easy to integrate**

AWS CodePipeline can easily be extended to adapt to your specific needs. You can use our pre-built plugins or your own custom plugins in any step of your release process. For example, you can pull your source code from GitHub, use your on-premises Jenkins build server, run load tests using a third-party service, or pass on deployment information to your custom operations dashboard.

AWS CodePipeline is similar to the Jenkins Pipeline which helps to have a visual view of the end-to-end delivery process.

So, in a CodePipeline, you will typically configure the following

* **Source Code Repository** – So your source code would need to be either in AWS CodeCommit or GitHub repository.
* **Build Service** – AWS CodeBuild details will be configured as part of the pipeline.
* **Deploy** – AWS CodeDeploy will be configured into the pipeline.
* During the deploy process to different environments if any approvals are needed they could be configured as well

So, if there is a code change by the developer the visual representation of Build and Deploy can be seen to be automated.

Source code repository configuration in AWS CodePipeline

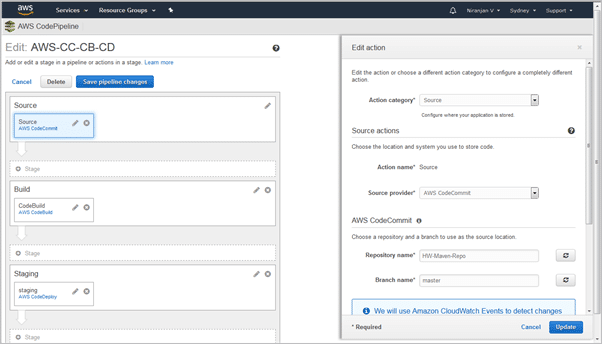
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image016-1.png)

Image 59: Code pipeline -Configuration

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Build configuration in AWS CodePipeline which uses Maven build

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image017.png)

Image 60: Code pipeline Maven build

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Deployment configuration in AWS CodePipeline

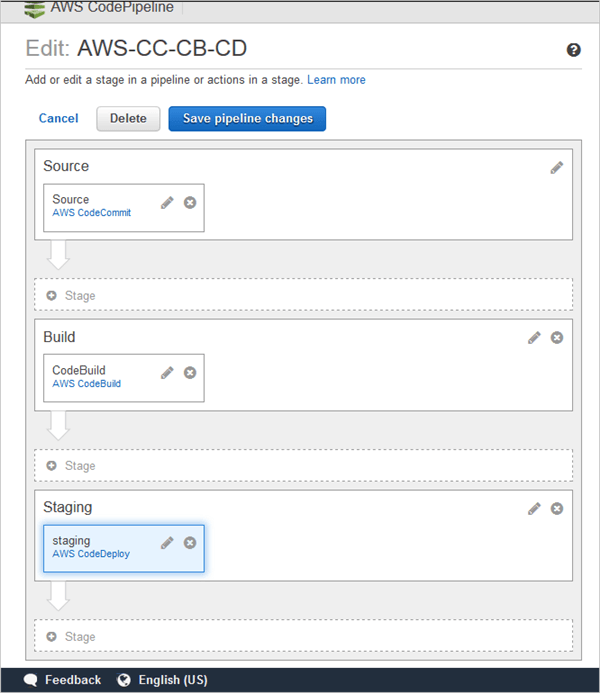
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image018.png)

Image 61: Code pipeline-Deployment

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

Complete Execution is seen in AWS CodePipeline

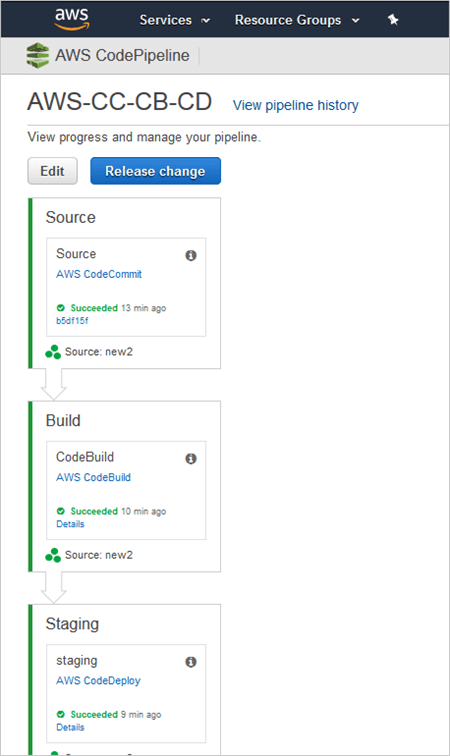
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2018/07/image019.png)

Image 62: Code pipeline Execution

Reference: <https://www.softwaretestinghelp.com/aws-devops-tools/>

# 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.

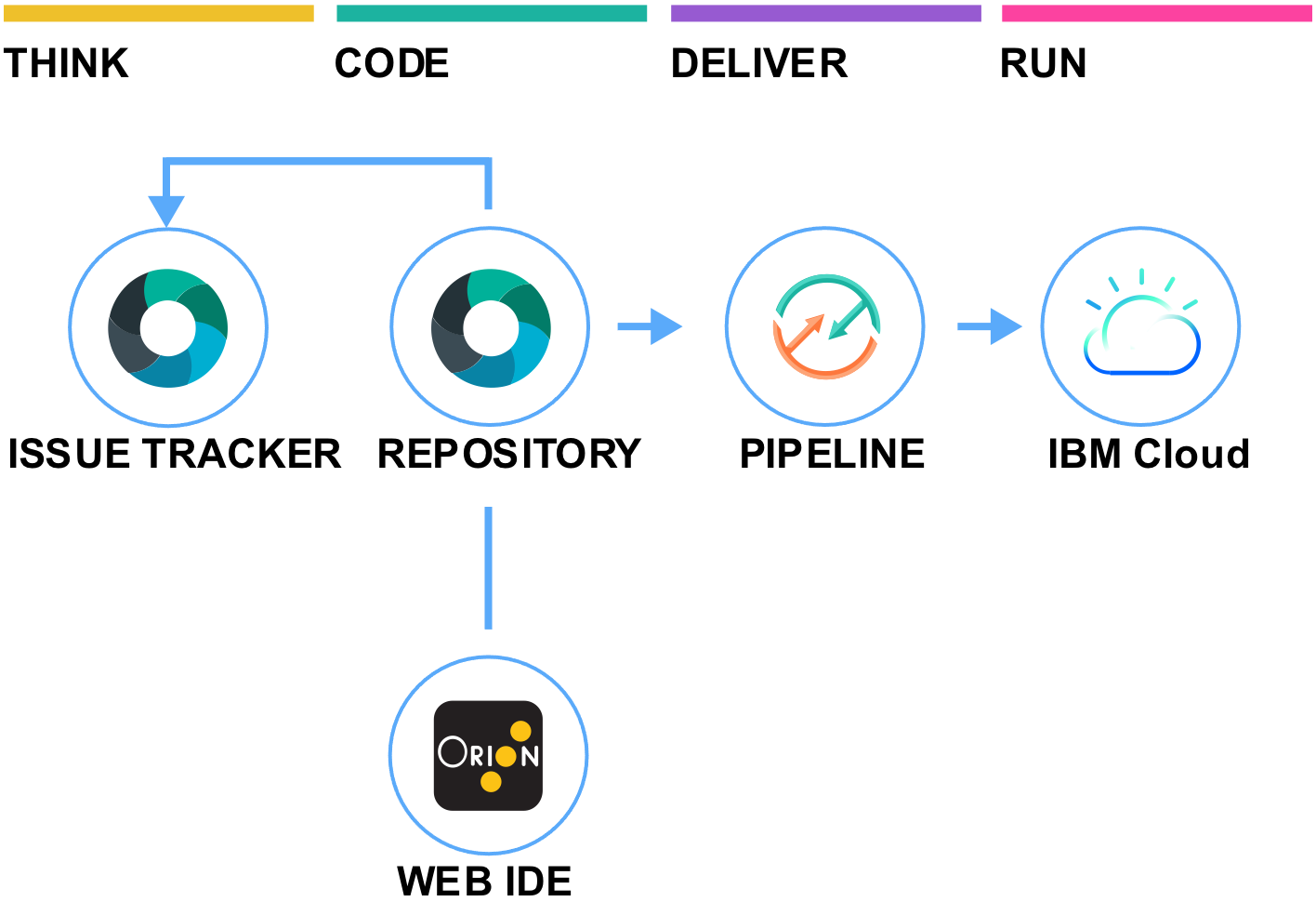
Some toolchain templates include tool integrations that are part of the Continuous Delivery service. If an instance of that service isn't already in your resource group, when you click **Create** to create the toolchain, the service is automatically added with the selected free Lite plan. For more information 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.

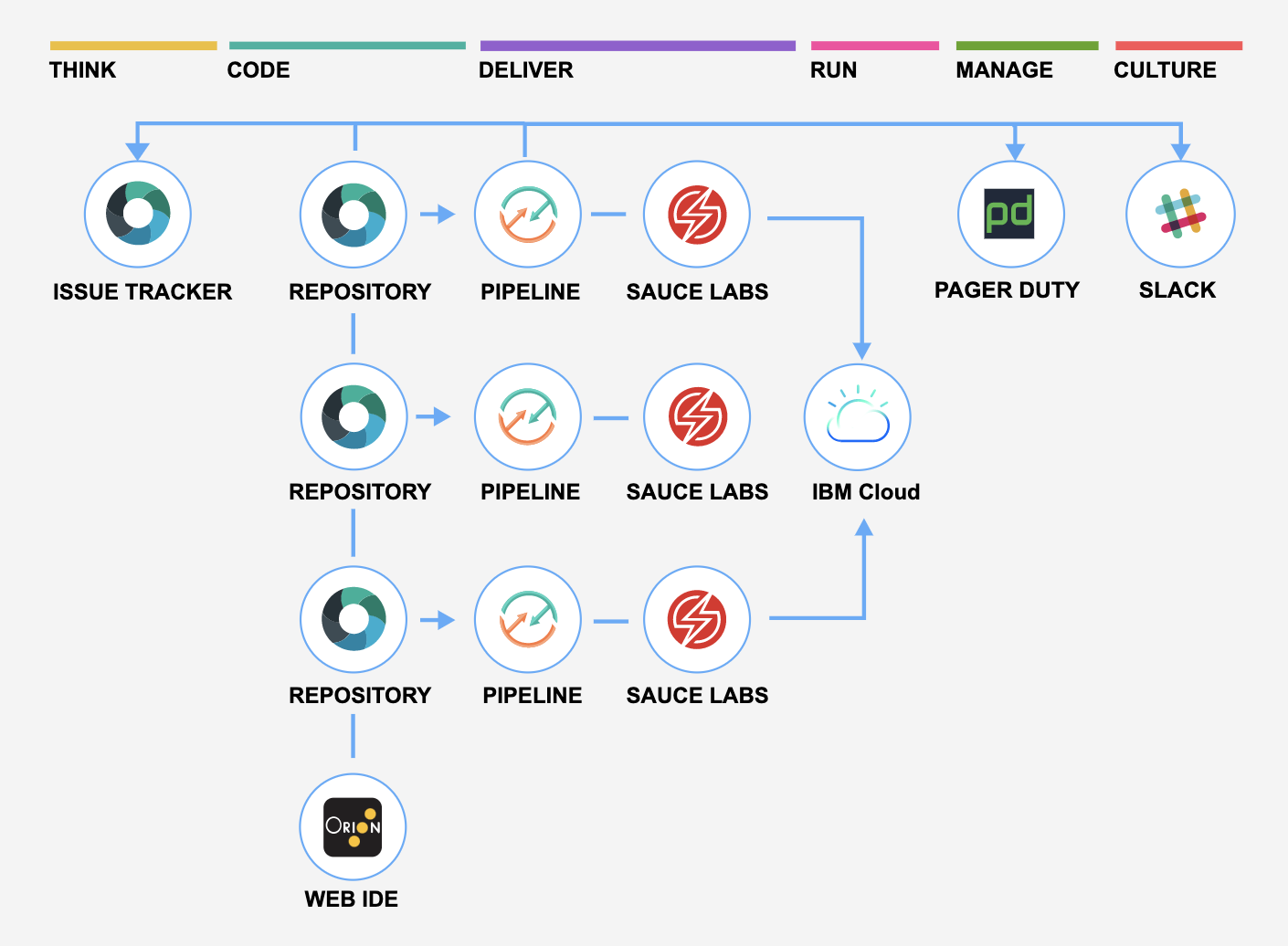
To see which toolchains and tool integrations are available, see [Toolchain availability, templates, and tutorials](https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-cd_about).

## 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.

Image 1. Sample Toolchain

Reference - <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.

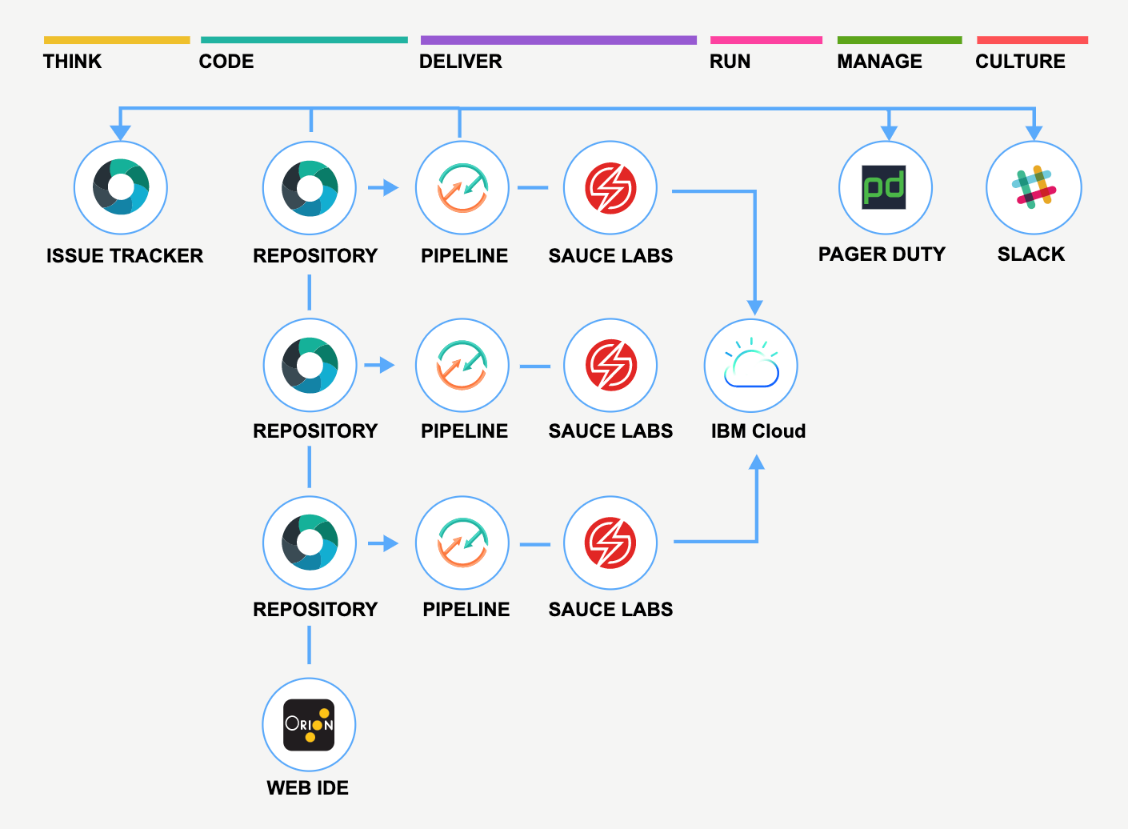


Image 2. Toolchain diagram

## 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**.

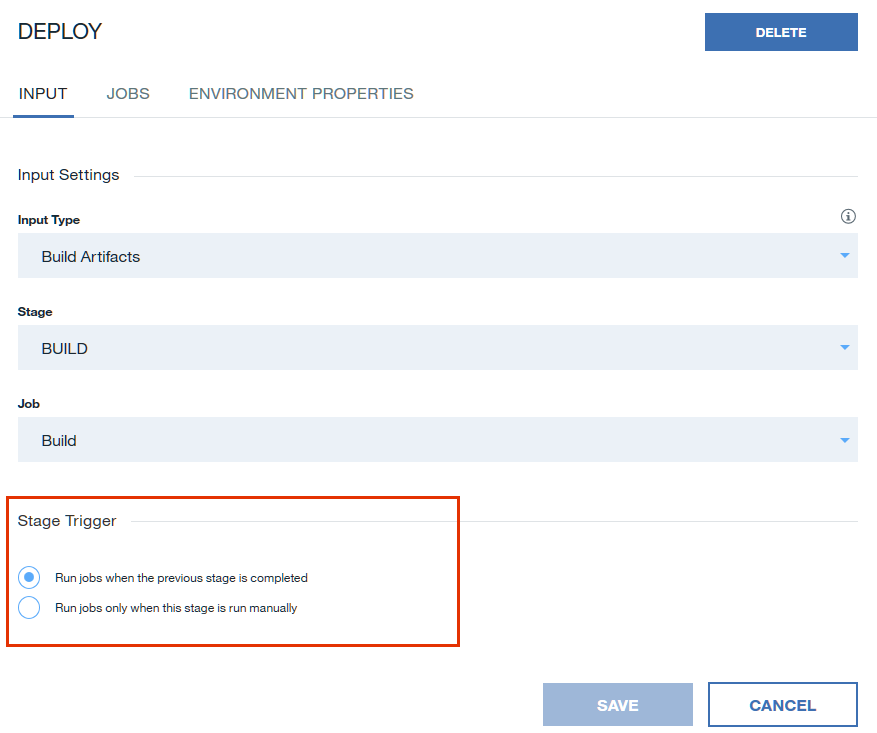


Image 3. Input tab

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.

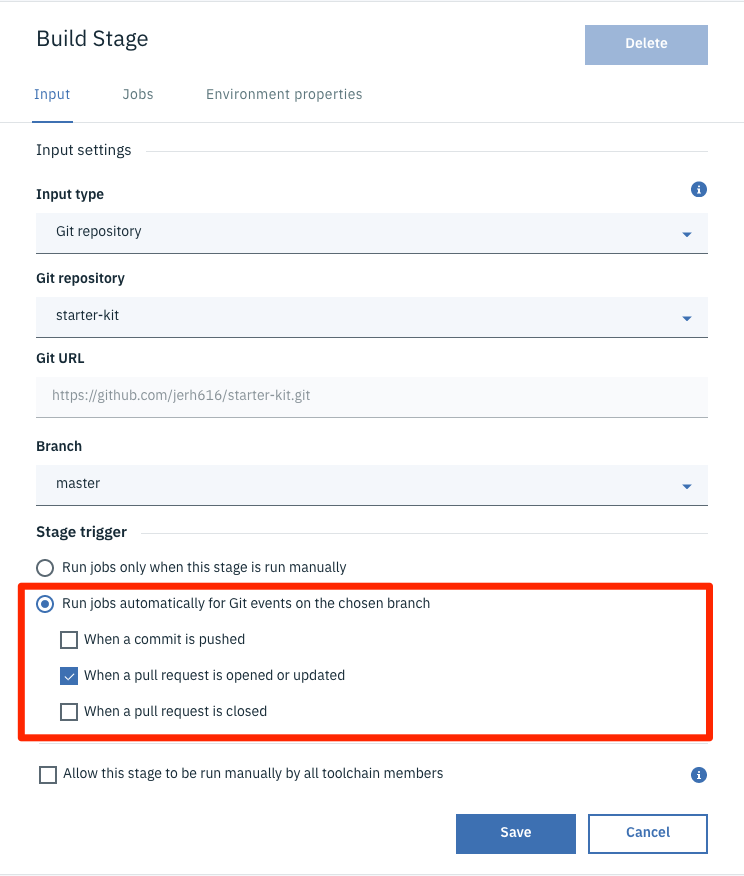


Image 4. Input tab triggers

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.

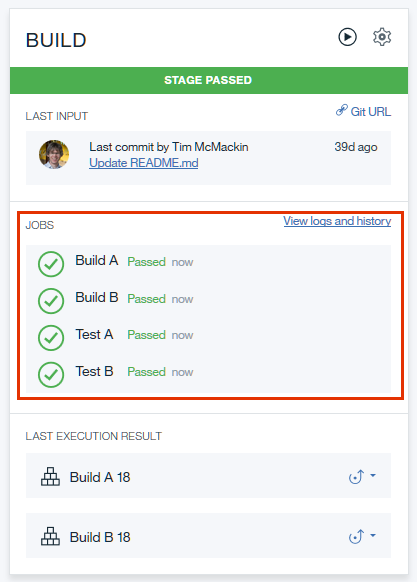


Image 5. Build and test jobs within a stage

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**.

Image 6. Pipeline overflow menu

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

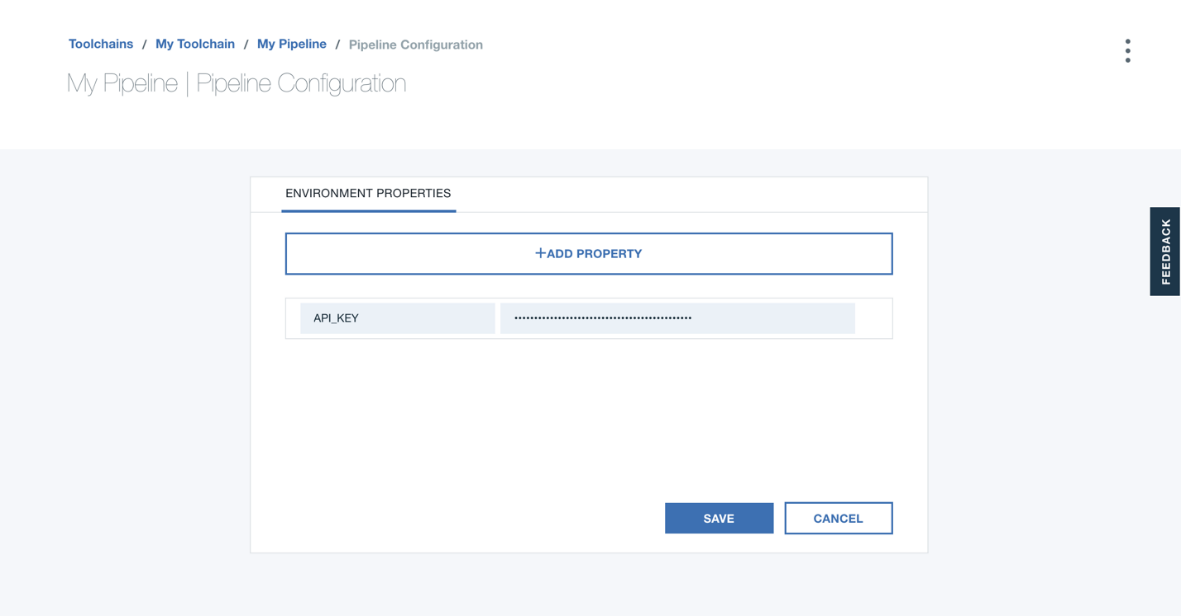
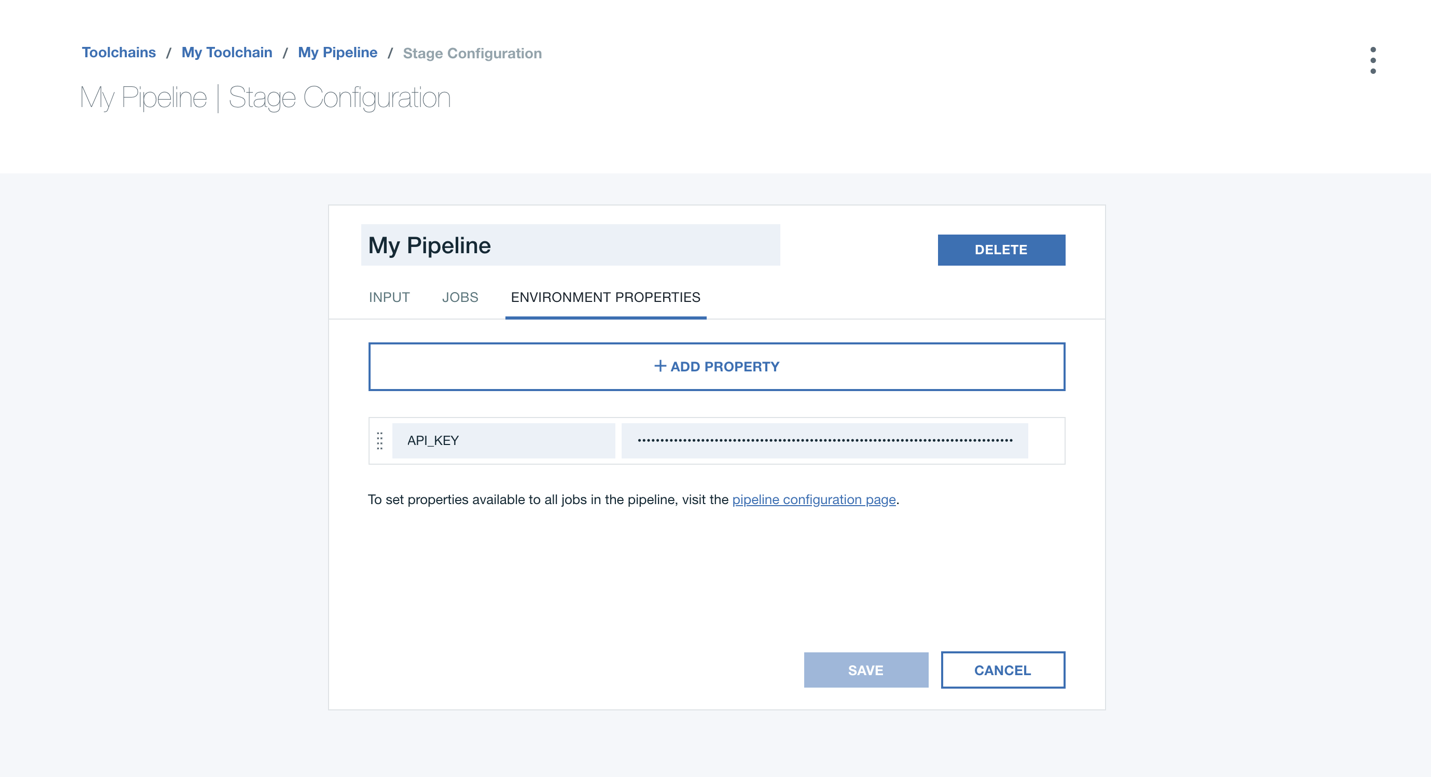


Image 7. Pipeline properties page

### **Stage properties**

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

Image 8. 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:

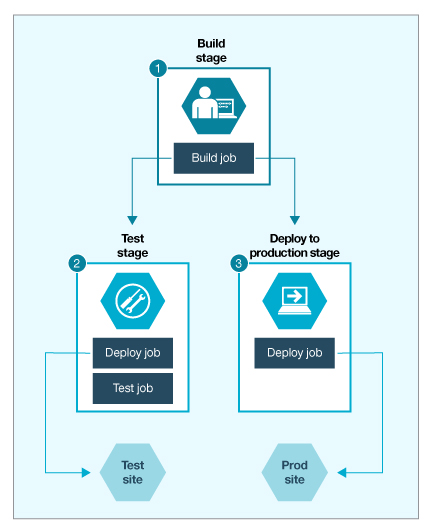


Image 9. Conceptual model of a three-stage pipeline

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.

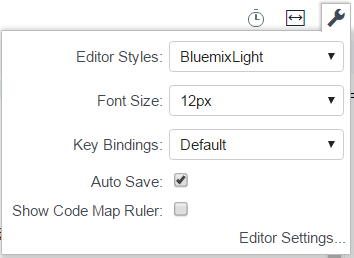


Image 10. Local Editor Settings

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.

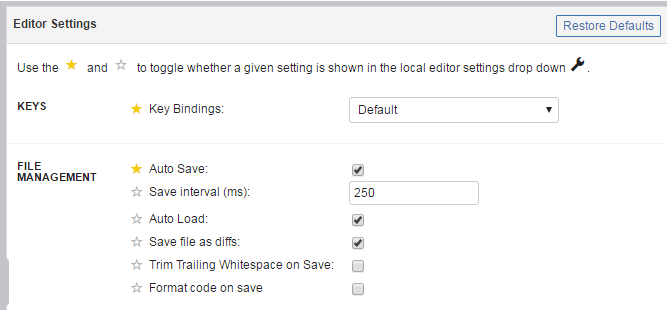


Image 11. Local Editor settings

## 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.

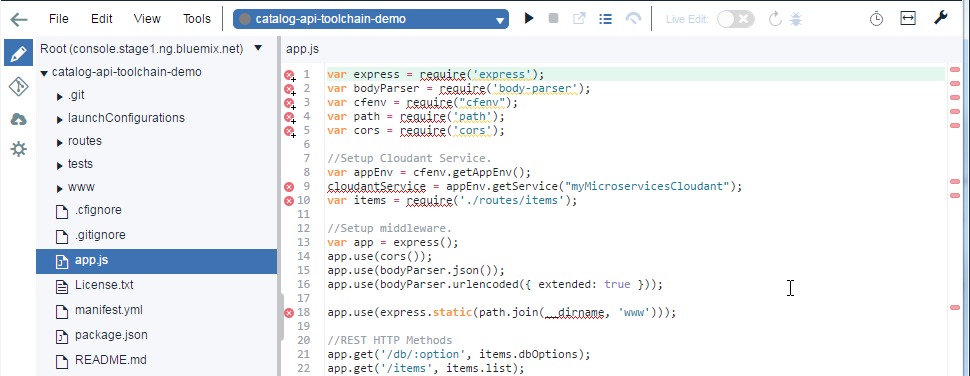


Image 12. Web IDE

### **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

# Introduction to Application Programming Interface (API)

## API Concept

An application programming interface, or API, enables companies to open up their applications’ data and functionality to external third-party developers, business partners, and internal departments within their companies. This allows services and products to communicate with each other and leverage each other’s data and functionality through a documented interface. Developers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services. API use has surged over the past decade, to the degree that many of the most popular web applications today would not be possible without APIs.

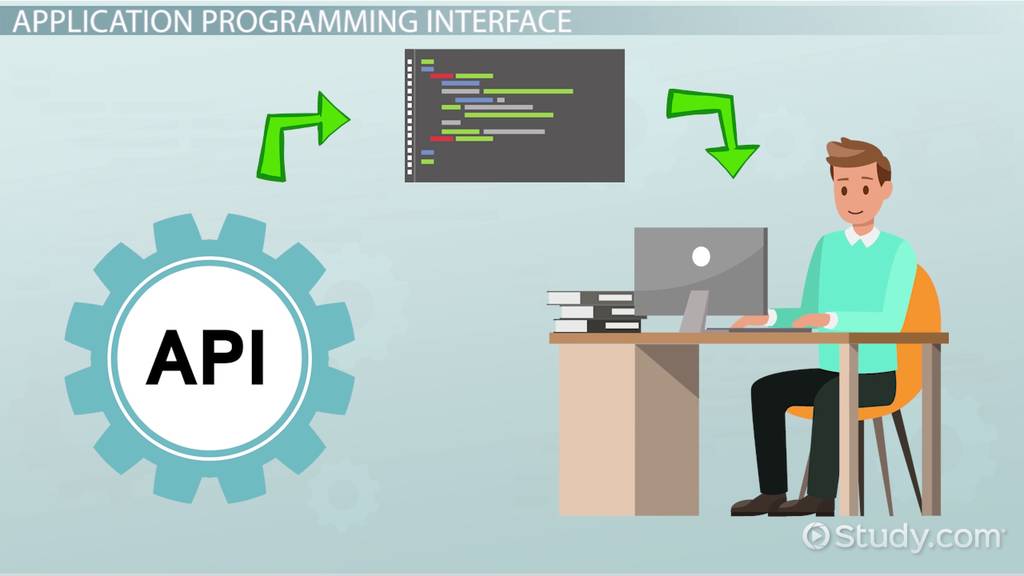


Image: API concept

Reference: <https://study.com/cimages/videopreview/kg7jeue1w3.jpg>

## How an API works

An API is a set of defined rules that explain how computers or applications communicate with one another. APIs sit between an application and the web server, acting as an intermediary layer that processes data transfer between systems.

Here’s how an API works:

1. A client application initiates an API call to retrieve information—also known as a request. This request is processed from an application to the web server via the API’s Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.
2. After receiving a valid request, the API makes a call to the external program or web server.
3. The server sends a response to the API with the requested information.
4. The API transfers the data to the initial requesting application.

While the data transfer will differ depending on the web service being used, this process of requests and response all happens through an API. Whereas a user interface is designed for use by humans, APIs are designed for use by a computer or application.



Image: How an API works

Reference: <https://i.pinimg.com/originals/f8/02/cc/f802cc6d8fbc9e3f4a9223eb1d965275.png>

APIs offer security by design because their position as middleman facilitates the abstraction of functionality between two systems—the API endpoint decouples the consuming application from the infrastructure providing the service. API calls usually include authorization credentials to reduce the risk of attacks on the server, and an API gateway can limit access to minimize security threats. Also, during the exchange, HTTP headers, cookies, or query string parameters provide additional security layers to the data.

For example, consider an API offered by a payment processing service. Customers can enter their card details on the frontend of an application for an ecommerce store. The payment processor doesn’t require access to the user’s bank account; the API creates a unique token for this transaction and includes it in the API call to the server. This ensures a higher level of security against potential hacking threats.

## Why we need APIs

Whether you’re managing existing tools or designing new ones, you can use an application programming interface to simplify the process. Some of the main benefits of APIs include the following:

### Improved collaboration:

The average enterprise uses almost 1,200 cloud applications (link resides outside of IBM), many of which are disconnected. APIs enable integration so that these platforms and apps can seamlessly communicate with one another. Through this integration, companies can automate workflows and improve workplace collaboration. Without APIs, many enterprises would lack connectivity and would suffer from informational silos that compromise productivity and performance.

### Easier innovation:

APIs offer flexibility, allowing companies to make connections with new business partners, offer new services to their existing market, and, ultimately, access new markets that can generate massive returns and drive digital transformation. For example, the company Stripe began as an API with just seven lines of code. The company has since partnered with many of the biggest enterprises in the world, diversified to offer loans and corporate cards, and was recently valued at USD 36 billion (link resides outside of IBM).

### Data monetization:

Many companies choose to offer APIs for free, at least initially, so that they can build an audience of developers around their brand and forge relationships with potential business partners. However, if the API grants access to valuable digital assets, you can monetize it by selling access (this is referred to as the API economy). When AccuWeather (link resides outside of IBM) launched its self-service developer portal to sell a wide range of API packages, it took just 10 months to attract 24,000 developers, selling 11,000 API keys and building a thriving community in the process.

### Added security:

As noted above, APIs create an added layer of protection between your data and a server. Developers can further strengthen API security by using tokens, signatures, and Transport Layer Security (TLS) encryption; by implementing API gateways to manage and authenticate traffic; and by practicing effective API management.

## Common API examples

Because APIs allow companies to open up access to their resources while maintaining security and control, they have become a valuable aspect of modern business. Here are some popular examples of application programming interfaces you may encounter:

### Universal logins:

A popular API example is the function that enables people to log in to websites by using their Facebook, Twitter, or Google profile login details. This convenient feature allows any website to leverage an API from one of the more popular services to quickly authenticate the user, saving them the time and hassle of setting up a new profile for every website service or new membership.

### Third-party payment processing:

For example, the now-ubiquitous "Pay with PayPal" function you see on ecommerce websites works through an API. This allows people to pay for products online without exposing any sensitive data or granting access to unauthorized individuals.

### Travel booking comparisons:

Travel booking sites aggregate thousands of flights, showcasing the cheapest options for every date and destination. This service is made possible through APIs that provide application users with access to the latest information about availability from hotels and airlines. With an autonomous exchange of data and requests, APIs dramatically reduce the time and effort involved in checking for available flights or accommodation.

### Google Maps:

One of the most common examples of a good API is the Google Maps service. In addition to the core APIs that display static or interactive maps, the app utilizes other APIs and features to provide users with directions or points of interest. Through geolocation and multiple data layers, you can communicate with the Maps API when plotting travel routes or tracking items on the move, such as a delivery vehicle.

### Twitter:

Each Tweet contains descriptive core attributes, including an author, a unique ID, a message, a timestamp when it was posted, and geolocation metadata. Twitter makes public Tweets and replies available to developers and allows developers to post Tweets via the company's API.

## Types of APIs

Nowadays, most application programming interfaces are web APIs that expose an application's data and functionality over the internet. Here are the four main types of web API:

### Open APIs

Open APIs are open source application programming interfaces you can access with the HTTP protocol. Also known as public APIs, they have defined API endpoints and request and response formats.

### Partner APIs

Partner APIs are application programming interfaces exposed to or by strategic business partners. Typically, developers can access these APIs in self-service mode through a public API developer portal. Still, they will need to complete an onboarding process and get login credentials to access partner APIs.

### Internal APIs

Internal APIs are application programming interfaces that remain hidden from external users. These private APIs aren't available for users outside of the company and are instead intended to improve productivity and communication across different internal development teams.

### Composite APIs

Composite APIs combine multiple data or service APIs. These services allow developers to access several endpoints in a single call. Composite APIs are useful in microservices architecture where performing a single task may require information from several sources.

## Types of API protocols

As the use of web APIs has increased, certain protocols have been developed to provide users with a set of defined rules that specifies the accepted data types and commands. In effect, these API protocols facilitate standardized information exchange:

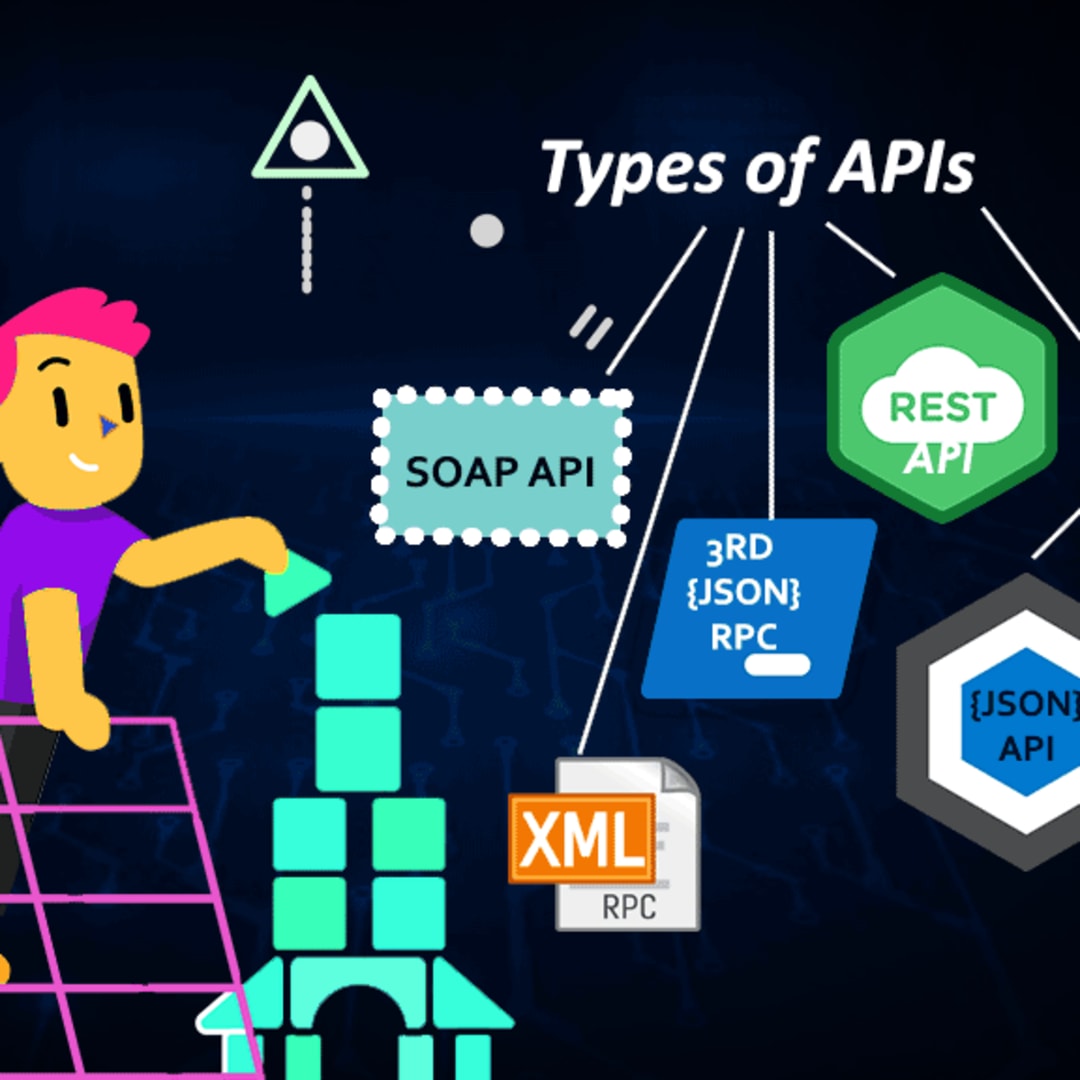


Image: Types of API protocol

Reference: [https://res.cloudinary.com/practicaldev/image/fetch/s--gtO8OvVY--/c\_imagga\_scale,f\_auto,fl\_progressive,h\_1080,q\_auto,w\_1080/https://thepracticaldev.s3.amazonaws.com/i/i2ktzzlyqsxtw9weaktc.png](https://res.cloudinary.com/practicaldev/image/fetch/s--gtO8OvVY--/c_imagga_scale,f_auto,fl_progressive,h_1080,q_auto,w_1080/https:/thepracticaldev.s3.amazonaws.com/i/i2ktzzlyqsxtw9weaktc.png)

### SOAP (Simple Object Access Protocol)

SOAP is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.

### XML-RPC

XML-RPC is a protocol that relies on a specific format of XML to transfer data, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.

### JSON-RPC

JSON-RPC is a protocol similar to XML-RPC, as they are both remote procedure calls (RPCs), but this one uses JSON instead of XML format to transfer data. Both protocols are simple. While calls may contain multiple parameters, they only expect one result.

### REST (Representational State Transfer)

REST is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It’s possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

# Introduction to REST API

An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST API is an API that conforms to the design principles of the REST, or representational state transfer architectural style. For this reason, REST APIs are sometimes referred to RESTful APIs.

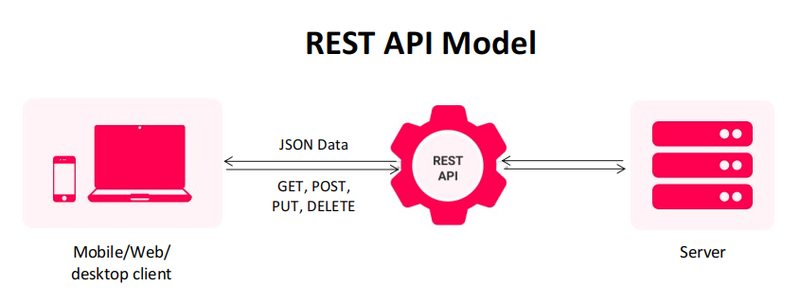


Image: What is REST API

Reference: <https://www.mindinventory.com/blog/wp-content/uploads/2021/09/rest-api-model-1.png>

First defined in 2000 by computer scientist Dr. Roy Fielding in his doctoral dissertation, REST provides a relatively high level of flexibility and freedom for developers. This flexibility is just one reason why REST APIs have emerged as a common method for connecting components and applications in a microservices architecture.

## REST design principles

At the most basic level, an API is a mechanism that enables an application or service to access a resource within another application or service. The application or service doing the accessing is called the client, and the application or service containing the resource is called the server.

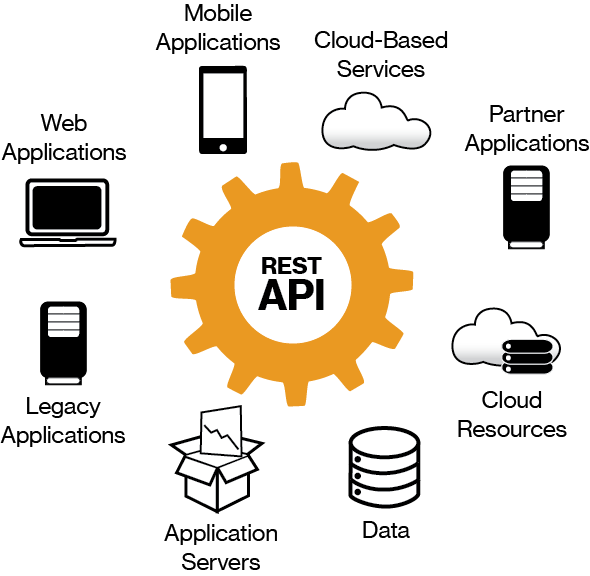


Image: REST design principles

Reference: <https://image.jimcdn.com/app/cms/image/transf/none/path/s1cbc048a561695b4/image/i49d13fd7b611bdcd/version/1470169946/image.png>

Some APIs, such as SOAP or XML-RPC, impose a strict framework on developers. But REST APIs can be developed using virtually any programming language and support a variety of data formats. The only requirement is that they align to the following six REST design principles - also known as architectural constraints:

### Uniform interface.

All API requests for the same resource should look the same, no matter where the request comes from. The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI). Resources shouldn’t be too large but should contain every piece of information that the client might need.

### Client-server decoupling.

In REST API design, client and server applications must be completely independent of each other. The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways. Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.

### Statelessness.

REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it. In other words, REST APIs do not require any server-side sessions. Server applications aren’t allowed to store any data related to a client request.

### Cacheability.

When possible, resources should be cacheable on the client or server side. Server responses also need to contain information about whether caching is allowed for the delivered resource. The goal is to improve performance on the client side, while increasing scalability on the server side.

### Layered system architecture.

In REST APIs, the calls and responses go through different layers. As a rule of thumb, don’t assume that the client and server applications connect directly to each other. There may be a number of different intermediaries in the communication loop. REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

### Code on demand (optional).

REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets). In these cases, the code should only run on-demand.

## How REST APIs work

REST APIs communicate via HTTP requests to perform standard database functions like creating, reading, updating, and deleting records (also known as CRUD) within a resource. For example, a REST API would use a GET request to retrieve a record, a POST request to create one, a PUT request to update a record, and a DELETE request to delete one. All HTTP methods can be used in API calls. A well-designed REST API is similar to a website running in a web browser with built-in HTTP functionality.

The state of a resource at any particular instant, or timestamp, is known as the resource representation. This information can be delivered to a client in virtually any format including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text. JSON is popular because it’s readable by both humans and machines—and it is programming language-agnostic.

Request headers and parameters are also important in REST API calls because they include important identifier information such as metadata, authorizations, uniform resource identifiers (URIs), caching, cookies and more. Request headers and response headers, along with conventional HTTP status codes, are used within well-designed REST APIs.

## REST API best practices

Although flexibility is a big advantage of REST API design, that same flexibility makes it easy to design an API that’s broken or performs poorly. For this reason, professional developers share best practices in REST API specifications.

The OpenAPI Specification (OAS) establishes an interface for describing an API in a way that allows any developer or application to discover it and fully understand its parameters and capabilities - available endpoints, allowed operations on each endpoint, operation parameters, authentication methods, and other information. The latest version, OAS3 (link resides outside IBM), includes with hands-on tools, such as the OpenAPI Generator, for generating API clients and server stubs in different programming languages.

Securing a REST API also starts with industry best practices, such as using hashing algorithms for password security and HTTPS for secure data transmission. An authorization framework like OAuth 2.0 (link resides outside IBM) can help limit the privileges of third-party applications. Using a timestamp in the HTTP header, an API can also reject any request that arrives after a certain time period. Parameter validation and JSON Web Tokens are other ways to ensure that only authorized clients can access the API.

## REST APIs and IBM Cloud

The benefits of REST APIs mean that they’ll continue to be an integral part of the software development process, especially as the demand for better customer experiences and more applications impacts business and IT operations.

When it comes to meeting these demands, a move toward greater automation will help. Ideally, it would start with small, measurably successful projects, which you can then scale and optimize for other processes and in other parts of your organization. Working with IBM, you’ll have access to AI-powered automation capabilities, including prebuilt workflows, to help accelerate innovation by making every process more intelligent.

IBM tools and services can help you address important issues surrounding APIs—including security, governance, and automation—as you continue modernizing your applications.

# API services in IBM and AWS Cloud

## IBM API Management on Cloud

Whether you use APIs as a proxy to forward requests to a backend resource and relay responses back to the calling application, or you use APIs as an assembly to transform and aggregate responses from external services before relaying data to the calling application, you can use the IBM® API Management on Cloud solution to quickly create, manage, and monitor these APIs.

By using the IBM API Management on Cloud solution, you can create, manage, and monitor a cloud-based environment from which you can create, promote, use, and track APIs.

The API Management environment provides the following functions:

* Manages the operations of the various servers in the API Management environment.
* Provides the tools to interface with the various servers.
* Provides analytic functions that collect and store information about APIs and API users.
* Processes and manages security protocols and stores relevant user and appliance authentication data.
* Provides assembly functions that enable APIs to integrate with various endpoints, such as databases or HTTP-based endpoints.

Built in components of the IBM API Management on Cloud solution provide an interface with an API Management environment:

* IBM API Management API Manager. A console that facilitates the creation, promotion, and tracking of APIs.
* IBM API Management Developer Portal. A customizable portal where you can socialize your APIs to encourage the development of new applications that extend the value of your core enterprise assets.

## Anatomy of the IBM API Management cloud

The IBM API Management cloud is a collection of servers that comprise an API Management installation, including the configuration information and metadata that they contain.

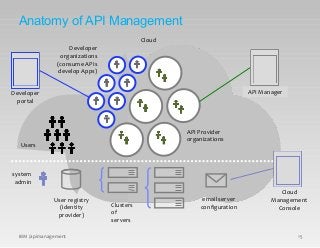


Image: Anatomy of the IBM API Management

Reference: <https://image.slidesharecdn.com/iag-apimanagementarchitectpresentation-150804045154-lva1-app6891/85/iag-api-management-architect-presentation-30-320.jpg?cb=1438664051>

The following diagram shows the relationship between the provider organizations, developer organizations, and users.

APIs are created and managed in the context of provider organizations.

Apps are created in the context of developer organizations. A developer organization belongs to a provider organization.

Users have an existence that is independent of an organization. A user can be a member of more than one provider or developer organization.

The cloud infrastructure is shared by all organizations, and managed independently of them.

## Inside organizations

The following diagram shows the relationship between the different types of organizations and environments.

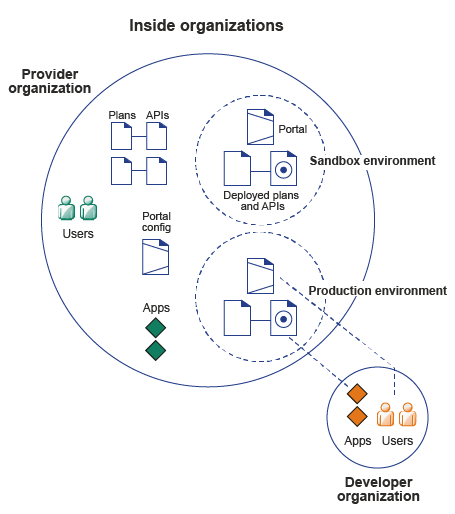


Image: Inside Organization

Reference: <https://www.ibm.com/docs/en/SSZFB2_3.0.1/com.ibm.apimgmt.overview.doc/orgs.gif>

An API is a set of functions that provide some business or technical capability and that can be called by applications by using a defined protocol.

An API is composed of resources or operations, which are offered in one of two styles:

* A REST API is structured according to the principles of Representational State Transfer, and typically uses the JSON data format. Some REST APIs might use XML or a combination of JSON or XML for structured data. APIs that work with content such as images, document content, or other media incorporate requests or responses that use other MIME types.
* A SOAP API is a web service that is exposed as an API.

An API is made visible on the Developer Portal by publishing one or more Plans that expose resources or operations that the API provides, and making those Plans visible either publicly, or to a selected set of developer organizations. APIs can have different revisions.

APIs and Plans are owned by a provider organization. APIs and Plans exist as authored artifacts that are visible in the API Manager. For developer organizations to be able to use APIs, APIs must be staged to an environment, and published to some or all organizations. An environment has an associated Developer Portal and runtime capability. For example, a simple provider organization might consist of a sandbox environment and a production environment.

Apps are registered to use APIs by using a selected Plan, which determines the API quota.

### Signing up to IBM API Management on Cloud

You can start to use IBM API Management on Cloud by registering for a free 30 day trial, which provides access to the API Manager user interface, after which you can complete your subscription.

### Purchasing IBM API Management on Cloud

At any time during your trial, or after your trial has expired, you can purchase IBM API Management on Cloud. You can also purchase without first subscribing to the trial.

### API Management on Cloud concepts

A high-level overview of IBM API Management on Cloud concepts.

### API Management on Cloud user roles

Defines the roles of various API Management on Cloud solution users.

### IBM API Management on Cloud software requirements

Ensure that you install the minimum prerequisite software for IBM API Management on Cloud.

### IBM API Management on Cloud support

You can obtain support by using online help, or, if necessary, by raising a support request with IBM.

### Glossary

You can use the IBM API Management glossary or the IBM Terminology website to check terms and definitions.

### Legal information

Notices, and terms and conditions for information centers.

## API Connect components

The API Connect components provide a unified user experience across the API lifecycle. Changes in one stage of the API lifecycle are automatically reflected in the other components of API Connect.

* Cloud Manager
* The developer toolkit
* API Manager
* API Gateways
* Runtime
* Developer Portal
* API Analytics
* Typical tasks per interface component
* API Connect server requirements

## Cloud Manager

The API Connect Cloud Manager component is used to manage the API Connect on-premises cloud. The Cloud Administrator uses this UI to:

* Define the cluster of Management servers, Gateway servers, and containers that are required in the cloud, and configure the topology. For information about Management servers and Gateway servers, see API Connect server requirements. For information about containers, see Runtime.
* Manage (modify, move, remove, restart, reboot) the servers in the cloud.
* Monitor the health of the cloud.
* Define and manage the provider organizations that develop APIs. (Assigned managers or owners of provider organizations can also complete this task.)
* Define additional cloud administrators, or set up users with roles that enable access to specific capabilities.
* Add user registries for authenticating users and securing APIs, and configure the secure transmission of data (for example, through websites).

## The developer toolkit

The developer toolkit provides the tools for modeling, developing, and testing APIs and LoopBack® applications. The developer toolkit includes a command line interface (CLI). It also incorporates LoopBack, an open source Node.js framework.

API developers use the API management functions in the API Manager or the CLI to create draft API definitions for REST and SOAP APIs, or for OAuth provider endpoints that are used for OAuth 2.0 authentication. The API definitions can be configured to add the API to a Product, add a policy assembly flow (to manipulate requests/responses), and to define security options and other settings. APIs can then be tested locally prior to publishing, to ensure they are defined and implemented correctly.

Using LoopBack, an API developer can create a Node.js application, connect to a data source such as a back-end database or a REST API to be consumed, and then expose the application as a REST API by creating a model definition. A LoopBack model defines the application data, validation rules, data access capabilities, and business logic for an API, and provides a REST API by default. This REST API can then be used by a REST API definition that was created using the API Manager or CLI and exposed to your users. The API and its associated application, which are implemented as a LoopBack project, must both be published to enable the project to be run. LoopBack projects can also be tested locally by creating a local runtime environment. The following diagram illustrates the LoopBack project architecture:

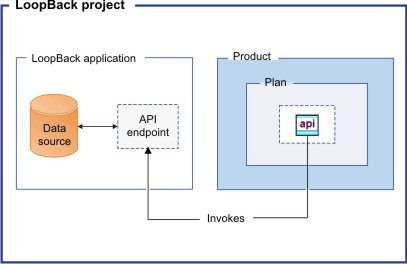


Image: The developer toolkit

Reference: <https://www.ibm.com/docs/en/SSMNED_v10/com.ibm.apic.overview.doc/overview_loopback_project.jpg>

Draft APIs (in their containing Products) that are created by using the API Manager, CLI, or LoopBack are published to Catalogs. Applications created using LoopBack are published to containers, from where they run when called. (For information about containers, see Runtime.)

The developer toolkit is installed locally, for offline API and application development. For more information about the developer toolkit, see Developing your APIs and applications. For more information about LoopBack, see LoopBack: The Node.js API Framework.

## API Manager

The API Manager provides a user interface that facilitates promotion and tracking of APIs that are packaged within Products and Plans. API providers can move the Products through their lifecycle, and manage the availability and visibility of APIs and Plans.

Catalogs and Spaces are created in the API Manager to act as staging targets through which APIs, Plans, and Products are published to consumer organizations. API providers can stage their Products to Catalogs or Spaces, and then publish them to make the APIs in those Products visible on a Developer Portal for external discovery.

To control access to the available API management functions, users in the provider organization can be set up in the API Manager UI with assigned roles and permissions. API providers can also use the UI to manage the consumer organizations that sign up to access their APIs and Plans. Developer communities can additionally be created as a way of grouping together a collection of consumer organizations to whom a particular set of Products and Plans can be made available.

The API Manager UI also includes functions to manage the security of the API environment, and provides access to analytics information about API invocation metrics within customizable dashboard views.

For more information about the API Manager, see Managing your APIs.

## API Gateways

Gateways enforce runtime policies to secure and control API traffic, provide the endpoints that expose APIs to the calling applications, and provide assembly functions that enable APIs to integrate with various endpoints. They also log and report all API interactions to the API Connect analytics engine, for real-time and historical analytics and reporting. The following Gateway is available for use in API Connect:

* The DataPower® Gateway is an enterprise API Gateway that is built for departments and cross-enterprise usage. This Gateway provides a comprehensive set of API policies for security, traffic management, mediation, acceleration, and non-HTTP protocol support. The DataPower Gateway is deployed on a virtual or physical DataPower appliance and supports multiple Catalogs per instance or cluster. The DataPower Gateway can handle enterprise level complex integration, and supports containers for flexible runtime management.

Your API Connect offering (or edition) can include a virtual DataPower Gateway, and support for a physical DataPower Gateway is also available, subject to certain conditions.

## Runtime

You can run applications and API implementations in API Connect in a containerized runtime.

### Containerized runtime

A containerized runtime environment provides a lightweight deployment location for APIs and applications. A container wraps an application in a complete file system that includes everything it needs to run, such as code, runtime, system tools, and system libraries. You can use Docker Swarm or Kubernetes containers to run your APIs and applications being managed by API Connect.

## Developer Portal

The Developer Portal provides a customizable self-service web-based portal to application developers to explore, discover, and subscribe to APIs.

When API providers publish APIs in the API Manager, those APIs are exposed in the Developer Portal for discovery and usage by application developers in consumer organizations. Application developers can access the Developer Portal UI to register their applications, discover APIs, use the required APIs in their applications (with access approval where necessary), and subsequently deploy those applications.

The Developer Portal provides additional features, such as forums, blogs, comments, and ratings, for socialization and collaboration. API consumers can also view analytics information about the APIs that are used by an application, or used within a consumer organization. For more information, see Developer Portal: socialize your APIs.

## API Analytics

API Connect provides the capability to filter, sort, and aggregate your API event data. This data is then presented within correlated charts, tables, and maps, to help you manage service levels, set quotas, establish controls, set up security policies, manage communities, and analyze trends. API analytics is built on the Kibana V5.5.1 open source analytics and visualization platform, which is designed to work with the Elasticsearch real-time distributed search and analytics engine.

## API Connect server requirements

From an on-premises cloud, you can create, promote, use, and track APIs. An on-premises cloud is composed of various appliances, where each appliance is a server of a specific type. The collection of servers defines your cloud and determines how to distribute the work of managing, analyzing, routing, and storing data.

Your on-premises cloud can be a combination of new and existing physical appliances and virtual appliances, or can be entirely composed of virtual appliances. The type and quantity of servers in an API Connect environment are determined by the individual needs of each enterprise, but the minimum requirement is one Management server, one Analytics server, one Gateway server, and one server to host the Developer Portal.

The API Connect on-premises cloud includes the following server types:

### Management server

Stores all of the cloud configuration, and controls communication between the other servers within API Connect. Manages the operations of the various servers in the API Connect cloud and provides the tools to interface with the various servers. The Cloud Manager and API Manager user interfaces run on the Management server.

### Analytics server

Provides analytic functions that collect and store information about APIs and API users.

### Gateway server

Processes and manages security protocols and stores relevant user and appliance authentication data. The Gateway server also provides assembly functions that enable APIs to integrate with various endpoints, such as databases or HTTP-based endpoints.

### Developer Portal server

Provides a customizable social developer portal with a full-featured content management system, and includes clustering capability. Enables API providers to build portals for their application developers, and provides the interface for application developers to discover APIs and subscribe to usage Plans contained in the published Products for use in their applications.

## API management tools on AWS

AWS offers a comprehensive platform for API management called Amazon API Gateway. Used across businesses and organizations, from enterprises to startups, API Gateway makes it easy to define, secure, deploy, share, and operate APIs at any scale. It also makes API monitoring simple and fast. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. API Gateway also offers a serverless developer portal that enables API publishers to easily connect with API subscribers, as well as easily monitor, manage, and update their APIs.

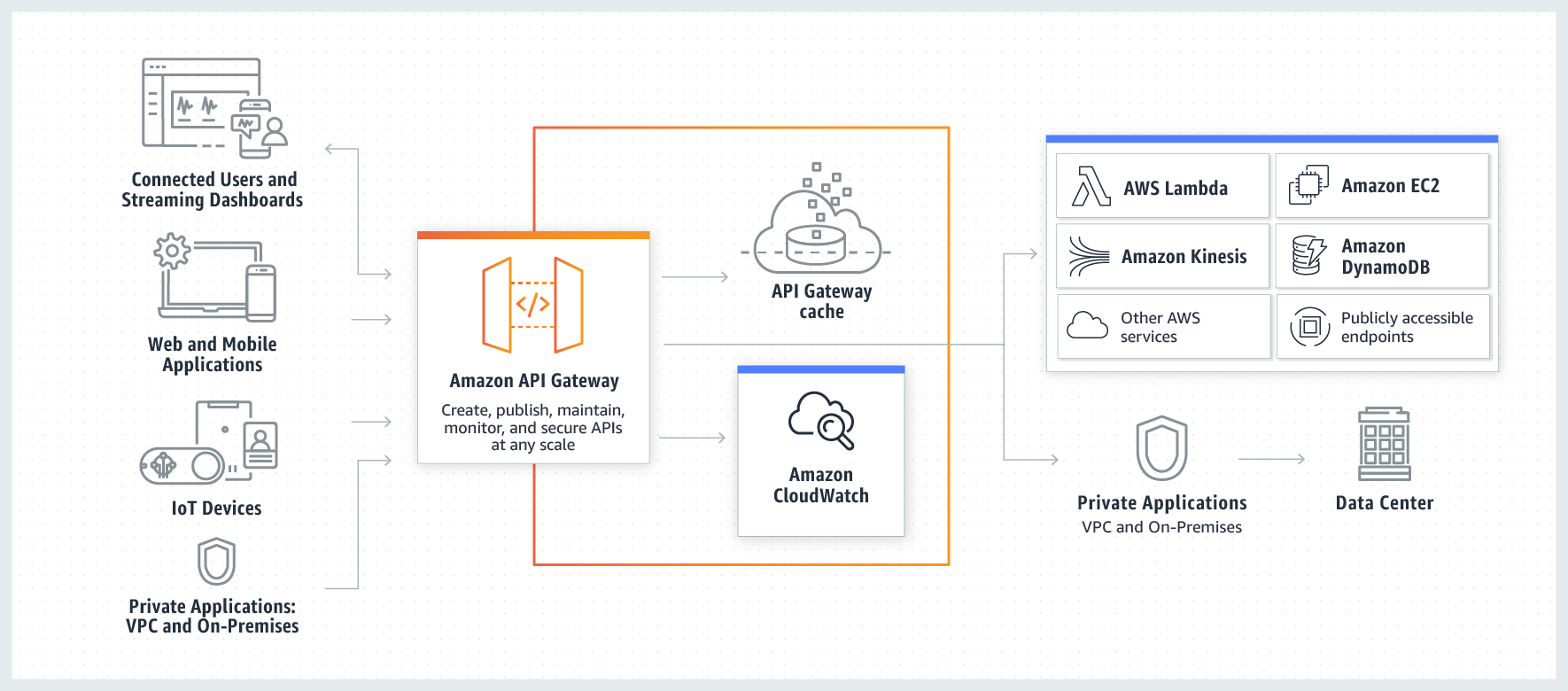


Image: API management tools on AWS

Reference: <https://d1.awsstatic.com/serverless/New-API-GW-Diagram.c9fc9835d2a9aa00ef90d0ddc4c6402a2536de0d.png>

# Amazon API Gateway benefits

### A SECURITY-FIRST APPROACH FOR API MANAGEMENT

Amazon API Gateway allows you to leverage the same technology AWS uses to run its own services, Signature Version 4. Using Signature Version 4 authentication, you can use Identity and Access Management (IAM) and access policies to authorize access to your APIs and all other AWS resources.

### API MANAGEMENT TOOLS FOR BUILDING AND DEPLOYING APIS

Amazon API Gateway can execute AWS Lambda code in your account, start AWS Step Functions state machines, or make calls to AWS Elastic Beanstalk, Amazon EC2, Amazon ECS, or web services outside of AWS with publicly accessible HTTP endpoints, like Docker. Using the Amazon API Gateway console, you can define your REST API and its associated resources and methods, manage your API lifecycle, generate your client SDKs, and view API metrics.

### API MANAGEMENT TOOLS FOR HANDS-OFF SCALING AND COMPLETE OPERATIONAL VISIBILITY

Amazon API Gateway handles any level of traffic received by an API, so you are free to focus on your business logic and services rather than maintaining infrastructure. Amazon API Gateway also provides you with a dashboard to visually monitor calls to the services. The Amazon API Gateway console is integrated with Amazon CloudWatch, so you have full visibility into backend performance metrics, such as API calls, latency, and error rates.

### API MANAGEMENT TOOLS FOR THIRD-PARTY ACCESS

Amazon API Gateway lets you create API keys, set fine-grained access permissions on each API key, and distribute them to third-party developers to access your APIs. You can also define plans that set throttling and request quota limits for each individual API key.

## Reference architectures for common API use cases

These reference architectures provide the architectural guidance you need to build an application that takes full advantage of Amazon API Gateway and the AWS Cloud.

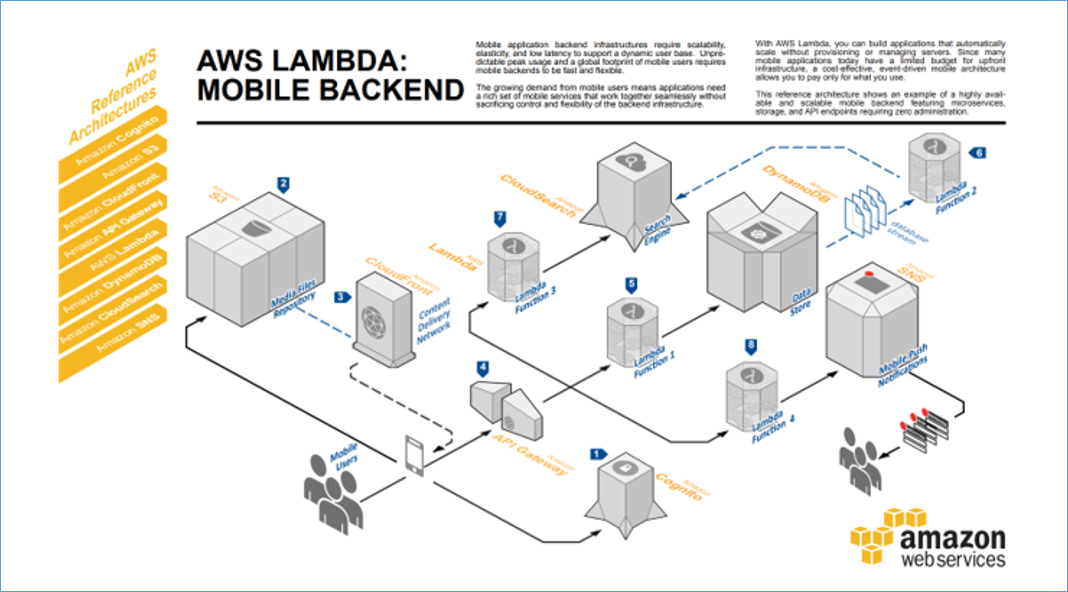


Image: Reference architectures for common API use cases

Reference: <https://d1.awsstatic.com/serverless/api_gateway_mobilebackend.5de49b6b47cf06a575ae588ba68ae0df6e578d0e.png>

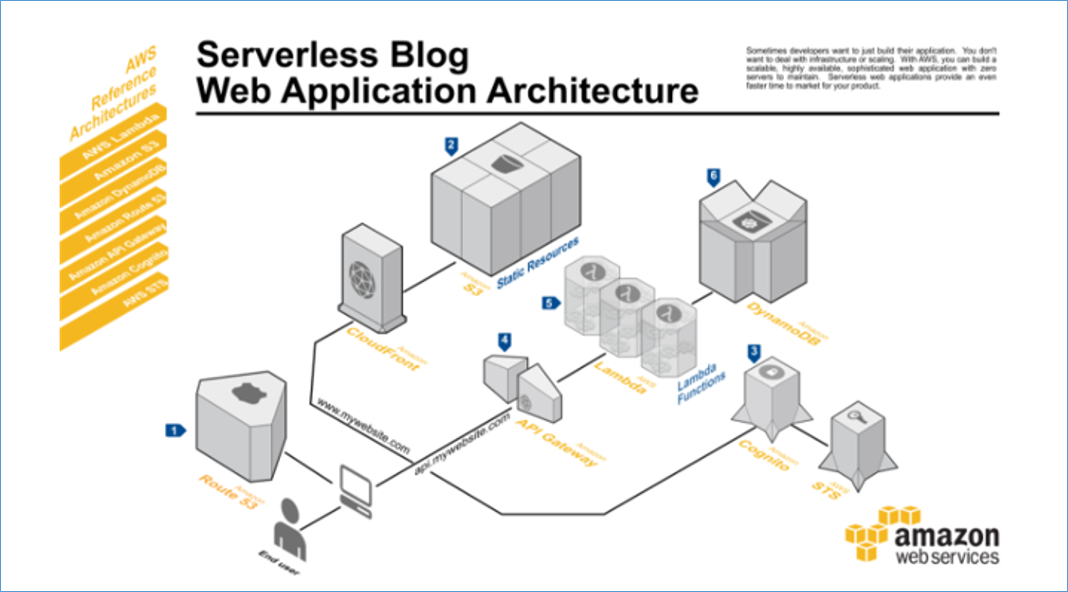


Image: Reference architectures for common API use cases

Reference: <https://d1.awsstatic.com/serverless/api_gateway_webapp.692a1575a8ed89c354218506a17591971658581e.png>

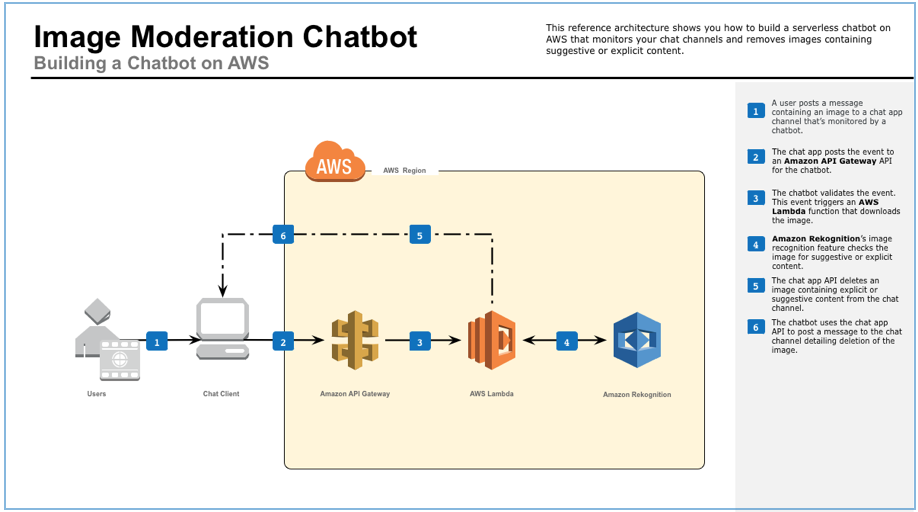


Image: Reference architectures for common API use cases

Reference: <https://d1.awsstatic.com/serverless/Image-Moderation-Chatbot.8e336d97319a2b12925b42ba107c642b7c758c63.png>

## API management on AWS: Customer case studies

From enterprises like Fox to government agencies like the UK Driver and Vehicle Licensing Agency, organizations increasingly leverage APIs across all industries and use cases. To learn more about more how companies use API management tools through Amazon API Gateway.



Image: API management tools on AWS

Reference: <https://d1.awsstatic.com/serverless/Fox-logo.bafd14abfae20da112b5ac01698769129fc9a2fd.png>

Fox Digital Entertainment Group uses a common API layer powered by Amazon API Gateway to build and deliver FOX NOW, an application that streams millions of hours of digital content to consumers across devices.



Image: API management tools on AWS

Reference: <https://d1.awsstatic.com/serverless/UK-driver-and-licensing-agency-logo.8411dc3ddc681afaded0332dedc2583c40a381e0.png>

The UK Driver and Vehicle Licensing Agency took an API-first approach to meet their exponentially increasing demand for information requests after switching to a digital means of excise-tax payment. By using Amazon API Gateway, the only thing they worry about is the code—the thing that creates the greatest value—and everything else is taken care of.

### AWS Case Study: FOX Digital Consumer Group

#### About Fox Digital Consumer Group

Fox Digital Consumer Group uses containerized microservices running on AWS to build and deliver FOX NOW, which streams millions of hours of digital content to consumers via web, mobile, and set-top devices.

Using a common API layer powered by Amazon API Gateway and Network Load Balancer, Fox can completely decouple the frontend of FOX NOW from the backend content system. This backend system is hosted on AWS and built by multiple teams using a microservices architecture. Every service runs in Docker containers managed by Amazon Elastic Container Service (Amazon ECS) and horizontally scaled via EC2 Auto Scaling groups.

The open nature of AWS and Amazon ECS allowed Fox to implement a service mesh for its microservices using linkerd, improving service-request routing and enabling inter-service communication.

### AWS Case Study: The UK Driver and Vehicle Licensing Agency

UK Driver and Vehicle Licensing Agency Supports Secure, Data-Driven Innovation Using AWS

UK Driver and Vehicle Licensing Agency (DVLA) holds more than 47 million driver records and collects about £6 billion each year in Vehicle Excise Duty. DVLA is an executive agency, sponsored by the UK Department for Transport. The information it collects helps improve road safety, reduce vehicle-related crime, support environmental initiatives, and limit vehicle tax evasion.

### The Challenge

#### The Public Cloud for the Public Good

With more than 50 million vehicle-licensing transactions per year and records on every vehicle in the United Kingdom, DVLA is the definitive source of truth for a vast amount of valuable public information. The organization has been gradually digitizing its services for more than 10 years using an API-based approach, and is continually looking for ways to modernize access to the valuable public data it maintains.

“Our data sets are classified as National Information Infrastructure, meaning they have the most potential to do good if opened up,” says Matt Lewis, chief architect at DVLA. “We want to enable the right people and organizations to build innovative, secure solutions that benefit the economy, increase road safety, and provide value to the public.”

This mission is helped by a government-wide initiative in the UK to move IT services to the public cloud. “Historically, we’ve run two government-owned data centers, but we’ve had to feed and water them ourselves,” says Lewis. “As we try to scale up to digitize services and make them more available, managing infrastructure becomes an expensive task for something that doesn’t really add value.” Amazon Web Services (AWS) has become the organization’s destination of choice for many of these services.

### Why Amazon Web Services

#### An API-First Approach

DVLA was experiencing massively increased demand for the information, rising from 600,000 to more than 70 million requests per month. This was partly due to the discontinuation of the paper “tax disc” once displayed on UK windscreens to communicate that the owner had paid vehicle-excise duty, requiring the public to rely on DVLA online services to identify the tax status of a vehicle.

The organization was also being contacted by companies wanting to build applications that would consume vehicle-related data. Making the data open and available to appropriate uses had significant potential to simplify processes, reduce cost and risk, and contribute to economic growth—but DVLA would need to handle transaction volumes anticipated to measure in billions per month. “We needed to move to an elastic platform that would allow us to scale up and down based on demand,” says Lewis.

DVLA already had services running on AWS and, based on positive experiences, decided to evaluate Amazon API Gateway as a way to provide controlled data access. “We stood up a working prototype in just a few days using Amazon API Gateway,” says Lewis. “The speed with which we were able to deliver it was unprecedented.” Because Amazon API Gateway provides a complete service architecture, the team didn’t have to deploy servers or install, manage, or maintain software.

“When we deployed on premises, we had fixed capacity and had to procure more servers if we went beyond a certain limit,” says Lewis. “That model changed with the cloud, but even then, we had to provision capacity, manage images, create hardened builds, and so on. Now, using Amazon API Gateway, we’ve moved into a serverless world where the only thing we have to worry about is the code—the thing that creates the greatest value—and everything else is taken care of.” DVLA is now able to provide scalable, secure, programmatic data access to police, local authorities, and third parties building value-added solutions.

DVLA is also experimenting with the AWS Lambda serverless compute service. “We started by identifying some use cases that would make sense, such as generating reports from API Gateway data that get delivered to our management and operational teams,” says Lewis. “We’re very interested in the serverless model because it is only triggered when an event happens and you only pay for it when it runs.”

### The Benefits

#### Managing Capacity and Security in a Hybrid World

Amazon API Gateway provides fully elastic scaling, but DVLA needed to control transaction throughput to avoid overtaxing on-premises systems to which the solution connects. “Some of the data and services we’re exposing have fixed resource capacities, so we use throttling to avoid affecting our operational services,” says Lewis. “API Gateway supports rate limits that enable us to control the number of requests in a graceful manner.” Other solutions would have required DVLA to build its own rate-limiting application.

Security was another top priority, because DVLA handles personally identifiable information (PII) and has responsibilities under the UK Data Protection Act to protect that data. “As we expand our APIs, some will be accessible by the public, and others will be controlled,” says Lewis. “By relying on AWS, we can implement levels of authentication and security that are appropriate to the data.”

As part of the move to distributed, “as-a-service” systems with rapid innovation cycles, DVLA is changing how it develops applications. “We are decomposing our applications into smaller, discrete components so we can choose the most appropriate technology,” says Lewis. “AWS introduced more than 1,000 new features in 2016 alone. We want to be running on right-sized, fine-grained components so we can bring in new capabilities as soon as they become available.” With this approach, DVLA can consume rather than manage commodity services, freeing developers to spend time on differentiated activities that create value for the citizens of the UK.

### Benefits of AWS

* Massive scalability to support billions of transactions per month
* Elastic scaling to meet widely variable demand
* Flexibility to throttle transaction volumes to support hybrid architectu
* Security and governance capabilities meeting rigorous government standards
* Reduced cost and increased agility with serverless computing

**Serverless Architectures**

A serverless architecture is a way to build and run applications and services without having to manage infrastructure. It is a software design pattern where applications are hosted by a third-party service, eliminating the need for server software and hardware management by the developer. Applications are broken up into individual functions that can be invoked and scaled individually.

**What is the cloud provider’s role in serverless?**

Under a serverless model, a cloud provider runs physical servers and dynamically allocates their resources on behalf of users who can deploy code straight into production.

Serverless computing offerings typically fall into two groups, Backend-as-a-Service (BaaS) and Function-as-a-Service (FaaS).

BaaS gives developers access to a variety of third-party services and apps. For instance, a cloud-provider may offer authentication services, extra encryption, cloud-accessible databases, and high-fidelity usage data. With BaaS, serverless functions are usually called through application programming interfaces (APIs).

More commonly, when developers refer to serverless, they’re talking about a FaaS model. Under FaaS, developers still write custom server-side logic, but it’s run-in containers fully managed by a cloud services provider.

The major public cloud providers all have one or more FaaS offerings. They include Amazon Web Services with AWS Lambda, Microsoft Azure with Azure Functions, Google Cloud with multiple offerings, and IBM Cloud with IBM Cloud Functions, among others.

Some organizations choose to operate their own FaaS environments using open source serverless platforms, including Red Hat® OpenShift® Serverless, which is built on the Knative project for Kubernetes.

**Why use serverless architectures?**

By using a serverless architecture, developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

**How does Serverless Architectures Work?**

Servers allow users to communicate with an application and access its business logic, but managing servers takes considerable time and resources. Teams have to maintain the server hardware, take care of software and security updates, and create backups in case of failure. By adopting serverless architecture, developers can offload these responsibilities to a third-party provider, enabling them to focus on writing application code.

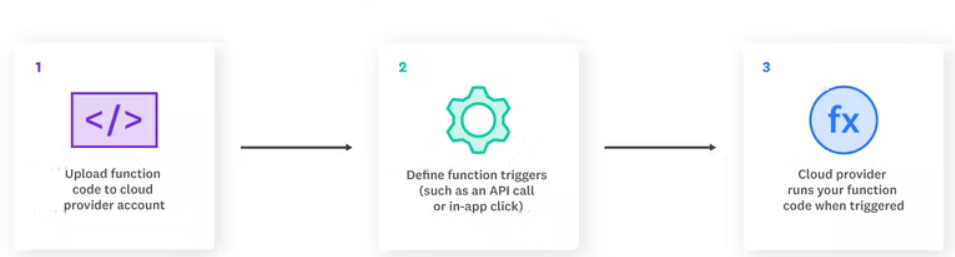


Image 1-Serverless functions work

Reference: https://www.datadoghq.com/knowledge-center/serverless-architecture/

Developers rely on serverless to execute specific functions. Because of this, cloud service providers offer Functions as a Service (FaaS). The steps to writing a function and execute it in a serverless way are following-

* **The developer writes a function.** This function often serves a specific purpose within the application code.
* **The developer defines an event.** The event is what triggers the cloud service provider to execute the function. A common type of event is an HTTP request.
* **The event is triggered.** If the defined event is an HTTP request, a user triggers the event with a click or similar action.
* **The function is executed.** The cloud service provider checks if an instance of the function is already running. If not, it starts a new one for the function.
* **The result is sent to the client.** The user sees the result of the executed function inside the application.

**Fundamental Concepts in Serverless Architecture**

Although serverless architecture eliminates the need for server management, there’s still a steep learning curve, especially if you’re chaining multiple functions together to create complex workflows in an application. It can therefore be helpful to familiarize yourself with these fundamental serverless terms:

* **Invocation**

A single function execution.

* **Duration**

The time it takes for a serverless function to execute.

* **Cold Start**

The latency that occurs when a function is triggered for the first time or after a period of inactivity.

* **Concurrency Limit**

The number of function instances that can run simultaneously in one region, as determined by the cloud provider. A function will be throttled if it exceeds this limit.

* **Timeout**

The amount of time that a cloud provider allows a function to run before terminating it. Most providers set a default timeout and a maximum timeout.

**Function-as-a-Service (FaaS)**

Function-as-a-Service (FaaS) is an event-driven computing execution model where developers write logic that is deployed in containers fully managed by a platform, then executed on demand.

In contrast to BaaS, FaaS affords a greater degree of control to the developers, who create custom apps rather than relying on a library of prewritten services.

Code is deployed into containers that are managed by a cloud provider. Specifically, these containers are:

* Stateless, making data integration simpler.
* Ephemeral, allowing them to be run for a very short time.
* Event-triggered, so they can run automatically when needed.
* Fully managed by a cloud provider, so that you only pay for what is needed, not always-on apps and servers.

Using FaaS, developers can call serverless apps through APIs which the FaaS provider handles through an API gateway.

# Popular Serverless Platforms

## Serverless is not only about a single service but a host of services that enable us to benefit from serverless architecture. There are quite a few cloud-services vendors available today that boast of a very mature catalog of serverless services: -

## Serverless Services on AWS

## The most important service in the catalog of serverless services on AWS is the FaaS service Lambda. AWS Lambda is a serverless compute service that lets us run our code without provisioning or managing servers. It allows us to write the Lambda functions in any of the popular languages like Node.js, Python, Go, and Java. We can simply upload our code as a ZIP file or container image.

## Hence, we get to use precise computation power for any scale of traffic. Moreover, AWS provides a host of other services for our serverless application. For instance, Amazon EventBridge for building event-driven applications. Amazon API Gateway to create, publish, maintain, monitor, and secure our APIs. Amazon S3 to store any amount of data with scalability and availability.

## Serverless Services on GCP

## Similarly, within the offerings of GCP, Cloud Functions is the key serverless product. Cloud Functions is a scalable pay-as-you-go FaaS to run our code with no server management. It automatically scales our application on the load. Further, it provides integrated monitoring, logging, and debugging. Moreover, it features built-in security at the role and per-function levels.

## Another key offering from GCP is the PaaS product App Engine. Cloud Functions is more suitable for simpler, standalone, and event-driven functions. But for more complex applications where we want to combine multiple functionalities, App Engine is a more suitable serverless platform. GCP also offers Cloud Run to deploy serverless applications packaged as containers on a Kubernetes cluster like GKE.

## Serverless Services on Azure

## On similar lines, Microsoft Azure offers a FaaS service Azure Functions that provides an event-driven serverless compute platform. We can also solve complex orchestration problems like stateful coordination using the Durable Functions extension. We can connect to a host of other services without any hard coding using triggers and bindings. Moreover, we can run our Functions on Kubernetes as well.

## As before, for more complex applications Azure also offers App Service. It allows us to build, deploy, and scale web applications on a fully managed serverless platform. We can create our application using a popular language like Node.js, Java, or Python. Moreover, we can meet rigorous enterprise-grade performance, security, and compliance requirements.

## Serverless on Kubernetes

## Kubernetes is an open-source orchestration system for automating deployment, scaling, and management of containerized workloads. Moreover, a dedicated infrastructure layer of service mesh like Istio can sit on top of the Kubernetes cluster. This can help us in solving a lot of common problems like service-to-service communication.

## It is natural to think about leveraging the combination of Istio on top of Kubernetes to host and manage a serverless environment. There can be a number of benefits of such a deployment model. For instance, it decouples us from many of the vendor-specific services. Moreover, it can provide a standard development framework for serverless applications.

## Serverless Architecture Use Cases

## Serverless architecture is best used to perform short-lived tasks and manage workloads that experience infrequent or unpredictable traffic. The main use cases for serverless include:

## Trigger-based tasks

## Any user activity that triggers an event or a series of events is a good candidate for serverless architecture. For instance, a user signing up on your website may trigger a database change, which may, in turn, trigger a welcome email. The backend work can be handled through a chain of serverless functions.

## Building RESTful APIs

## You can leverage Amazon API Gateway with serverless functions to build RESTful APIs that scale with demand.

## Asynchronous processing

## Serverless functions can handle behind-the-scenes application tasks, such as rendering product information or transcoding videos after upload, without interrupting the flow of the application or adding user-facing latency.

## Security checks

## When you spin up a new container, a function can be invoked to scan the instance for misconfigurations or vulnerabilities. Functions can also be used as a more secure option for SSH verification and two-factor authentication.

## Continuous Integration (CI) and Continuous Delivery (CD)

## Serverless architectures can automate many of the stages in your CI/CD pipelines. For example, code commits can trigger a function to create a build, and pull requests can trigger automated tests.

## 

Image 2-Use case for Serverless architecture

Reference: <https://www.datadoghq.com/knowledge-center/serverless-architecture/>

## Advantages of using serverless architecture

## Decomposing drives better observability

## With serverless, you break down applications into smaller and smaller pieces, known as decomposition. By doing so, you’ll gain better observability across the application.

## With smaller pieces, the knowledge necessary to make changes or create fixes is smaller.

## Serverless is event-based

## Serverless uses an event-based system versus stream-based

## With event-based architecture, each subpart of the application is independent. Events trigger one another. In stream-based, there are connections to each service. If there is a failure, it just impacts that event, not the entire log.

## Faster deployments, greater flexibility, and accelerated innovation

## Speed is often a contributing factor in choosing to use a serverless architecture. You can rapidly deploy apps in hours because there’s no infrastructure construction to weigh you down. With faster deployments also comes ease in scalability.

## By using such an agile architecture, you can be very flexible in your releases. Because it’s a quicker process, you can accelerate innovation.

## This flexibility is especially valuable in situations where pivoting is urgent. These types of scenarios are playing out all over the world in response to the pandemic. Organizations have to change their focus to meet emerging needs. This could be internal with a move to remote work. Another example is the adoption of customer-facing applications like those of retailers and restaurants.

## Reducing architecture costs

## Being serverless, an organization is essentially outsourcing server and database management. You are no longer responsible for the huge investments required for internal architecture administration. Ultimately your use case will define how much you can save.

## Focusing more on UX

## If your applications have end users, which they probably do, they have high expectations around digital experiences. If architecture is no longer a concern, it leaves more time to work on the user experience (UX). You can’t afford to not invest in the user interface, so serverless can provide you with a way to reallocate resources.

## Limitations of serverless architecture

## Long-running application inefficiencies

## Running workloads, which are long-running, could be more costly on serverless. Using a dedicated server is often more efficient.

## Third-party dependency

## Serverless architecture requires you to be reliant on your provider. You don’t have full control, and changes may impact you without notice. The platform’s availability is subject to its terms.

## Cold starts

## A “cold start” occurs when a platform must initiate internal resources. It may take some time for your serverless architecture to handle that first function request. You can avoid a “cold start” by ensuring the function remains in an active state. You do this by sending requests to it periodically.

# Container applications

## What are containers?

Containers are executable units of software in which application code is packaged, along with its libraries and dependencies, in common ways so that it can be run anywhere, whether it be on desktop, traditional IT, or the cloud.

To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS (in the case of the Linux kernel, namely the namespaces and cgroups primitives) are leveraged to both isolate processes and control the amount of CPU, memory, and disk that those processes have access to.

Containers are small, fast, and portable because unlike a virtual machine, containers do not need include a guest OS in every instance and can, instead, simply leverage the features and resources of the host OS.

Containers first appeared decades ago with versions like FreeBSD Jails and AIX Workload Partitions, but most modern developers remember 2013 as the start of the modern container era with the introduction of Docker.

## Benefits of containers

The primary advantage of containers, especially compared to a VM, is providing a level of abstraction that makes them lightweight and portable.

**Lightweight:** Containers share the machine OS kernel, eliminating the need for a full OS instance per application and making container files small and easy on resources. Their smaller size, especially compared to virtual machines, means they can spin up quickly and better support cloud-native applications that scale horizontally.

**Portable and platform independent:** Containers carry all their dependencies with them, meaning that software can be written once and then run without needing to be re-configured across laptops, cloud, and on-premises computing environments.

**Supports modern development and architecture:** Due to a combination of their deployment portability/consistency across platforms and their small size, containers are an ideal fit for modern development and application patterns—such as DevOps, serverless, and microservices—that are built are regular code deployments in small increments.

**Improves utilization:** Like VMs before them, containers enable developers and operators to improve CPU and memory utilization of physical machines. Where containers go even further is that because they also enable microservice architectures, application components can be deployed and scaled more granularly, an attractive alternative to having to scale up an entire monolithic application because a single component is struggling with load.

## Use cases for containers

Containers are becoming increasingly prominent, especially in cloud environments. Many organizations are even considering containers as a replacement of VMs as the general-purpose compute platform for their applications and workloads. But within that very broad scope, there are key use cases where containers are especially relevant.

**Microservices:** Containers are small and lightweight, which makes them a good match for microservice architectures where applications are constructed of many, loosely coupled and independently deployable smaller services.

**DevOps:** The combination of microservices as an architecture and containers as a platform is a common foundation for many teams that embrace DevOps as the way they build, ship and run software.

**Hybrid, multi-cloud:** Because containers can run consistently anywhere, across laptop, on-premises and cloud environments, they are an ideal underlying architecture for hybrid cloud and multicloud scenarios where organizations find themselves operating across a mix of multiple public clouds in combination with their own data center.

**Application modernizing and migration:** One of the most common approaches to application modernization starts by containerizing them so that they can be migrated to the cloud.

## Containerization

Software needs to be designed and packaged differently in order to take advantage of containers—a process commonly referred to as containerization.

When containerizing an application, the process includes packaging an application with its relevant environment variables, configuration files, libraries, and software dependencies. The result is a container image that can then be run on a container platform.

## Container orchestration with Kubernetes

As companies began embracing containers—often as part of modern, cloud-native architectures—the simplicity of the individual container began colliding with the complexity of managing hundreds (even thousands) of containers across a distributed system.

To address this challenge, container orchestration emerged as a way managing large volumes of containers throughout their lifecycle, including:

* Provisioning
* Redundancy
* Health monitoring
* Resource allocation
* Scaling and load balancing
* Moving between physical hosts

While many container orchestration platforms (such as Apache Mesos, Nomad, and Docker Swarm) were created to help address these challenges, Kubernetes, an open source project introduced by Google in 2014, quickly became the most popular container orchestration platform, and it is the one the majority of the industry has standardized on.

Kubernetes enables developers and operators to declare a desired state of their overall container environment through YAML files, and then Kubernetes does all the hard work establishing and maintaining that state, with activities that include deploying a specified number of instances of a given application or workload, rebooting that application if it fails, load balancing, auto-scaling, zero downtime deployments and more.

## Docker

Docker is a container management service. The keywords of Docker are developed, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

### Features of Docker

* Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.
* With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
* You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.
* Since Docker containers are pretty lightweight, they are very easily scalable.

### Components of Docker

Docker has the following components

* Docker for Mac − It allows one to run Docker containers on the Mac OS.
* Docker for Linux − It allows one to run Docker containers on the Linux OS.
* Docker for Windows − It allows one to run Docker containers on the Windows OS.
* Docker Engine − It is used for building Docker images and creating Docker containers.
* Docker Hub − This is the registry which is used to host various Docker images.
* Docker Compose − This is used to define applications using multiple Docker containers.

The official site for Docker is https://www.docker.com/ The site has all information and documentation about the Docker software.

### Docker – Hub

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub.

The official site for Docker hub is − <https://www.docker.com/community-edition#/add_ons>

### Docker – Images

In Docker, everything is based on Images. An image is a combination of a file system and parameters.

### Docker – Containers

Containers are instances of Docker images that can be run using the Docker run command. The basic purpose of Docker is to run containers.

### Docker – Container Lifecycle

The following illustration explains the entire lifecycle of a Docker container.

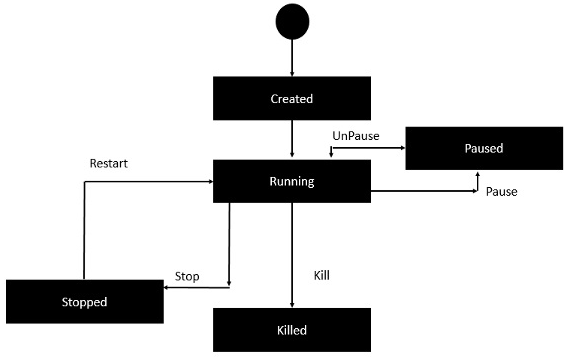


Image : Docker-Container Life Cycle

Reference: <https://www.tutorialspoint.com/docker/docker_working_with_containers.htm>

### Docker – Architecture

The following image shows the standard and traditional architecture of virtualization.

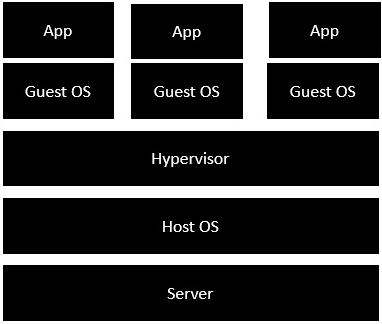


Image : Architecture of virtualization

Reference: <https://www.tutorialspoint.com/docker/docker_architecture.htm>

* The server is the physical server that is used to host multiple virtual machines.
* The Host OS is the base machine such as Linux or Windows.
* The Hypervisor is either VMWare or Windows Hyper V that is used to host virtual machines.
* You would then install multiple operating systems as virtual machines on top of the existing hypervisor as Guest OS.
* You would then host your applications on top of each Guest OS.

The following image shows the new generation of virtualization that is enabled via Dockers. Let’s have a look at the various layers.

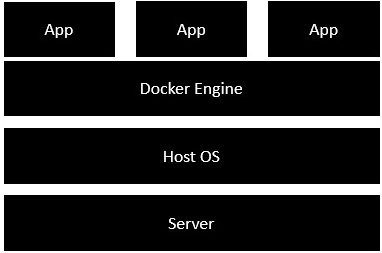


Image : Architecture of Docker

Reference: <https://www.tutorialspoint.com/docker/docker_architecture.htm>

* The server is the physical server that is used to host multiple virtual machines. So this layer remains the same.
* The Host OS is the base machine such as Linux or Windows. So this layer remains the same.
* Now comes the new generation which is the Docker engine. This is used to run the operating system which earlier used to be virtual machines as Docker containers.
* All of the Apps now run as Docker containers.

The clear advantage in this architecture is that you don’t need to have extra hardware for Guest OS. Everything works as Docker containers.

## Kubernetes

Kubernetes in an open source container management tool hosted by Cloud Native Computing Foundation (CNCF). This is also known as the enhanced version of Borg which was developed at Google to manage both long running processes and batch jobs, which was earlier handled by separate systems.

Kubernetes comes with a capability of automating deployment, scaling of application, and operations of application containers across clusters. It is capable of creating container centric infrastructure.

### Features of Kubernetes

Following are some of the important features of Kubernetes.

* Continues development, integration and deployment
* Containerized infrastructure
* Application-centric management
* Auto-scalable infrastructure
* Environment consistency across development testing and production
* Loosely coupled infrastructure, where each component can act as a separate unit
* Higher density of resource utilization
* Predictable infrastructure which is going to be created

One of the key components of Kubernetes is, it can run application on clusters of physical and virtual machine infrastructure. It also has the capability to run applications on cloud. It helps in moving from host-centric infrastructure to container-centric infrastructure.

## Kubernetes – Architecture

### Kubernetes - Cluster Architecture

As seen in the following diagram, Kubernetes follows client-server architecture. Wherein, we have master installed on one machine and the node on separate Linux machines.

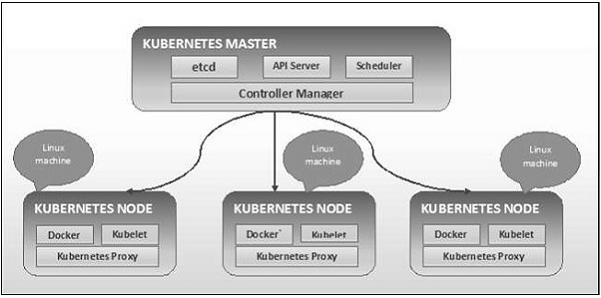


Image : Architecture of Kubernetes

Reference: <https://www.tutorialspoint.com/kubernetes/kubernetes_architecture.htm>

### Kubernetes - Master Machine Components

Following are the components of Kubernetes Master Machine.

**etcd**

It stores the configuration information which can be used by each of the nodes in the cluster. It is a high availability key value store that can be distributed among multiple nodes. It is accessible only by Kubernetes API server as it may have some sensitive information. It is a distributed key value Store which is accessible to all.

**API Server**

Kubernetes is an API server which provides all the operation on cluster using the API. API server implements an interface, which means different tools and libraries can readily communicate with it. Kubeconfig is a package along with the server side tools that can be used for communication. It exposes Kubernetes API.

**Controller Manager**

This component is responsible for most of the collectors that regulates the state of cluster and performs a task. In general, it can be considered as a daemon which runs in nonterminating loop and is responsible for collecting and sending information to API server. It works toward getting the shared state of cluster and then make changes to bring the current status of the server to the desired state. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.

**Scheduler**

This is one of the key components of Kubernetes master. It is a service in master responsible for distributing the workload. It is responsible for tracking utilization of working load on cluster nodes and then placing the workload on which resources are available and accept the workload. In other words, this is the mechanism responsible for allocating pods to available nodes. The scheduler is responsible for workload utilization and allocating pod to new node.

### Kubernetes - Node Components

Following are the key components of Node server which are necessary to communicate with Kubernetes master.

**Docker**

The first requirement of each node is Docker which helps in running the encapsulated application containers in a relatively isolated but lightweight operating environment.

**Kubelet Service**

This is a small service in each node responsible for relaying information to and from control plane service. It interacts with etcd store to read configuration details and wright values. This communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

**Kubernetes Proxy Service**

This is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers’ health checkup, etc.

### Kubernetes - Master and Node Structure

The following illustrations show the structure of Kubernetes Master and Node.

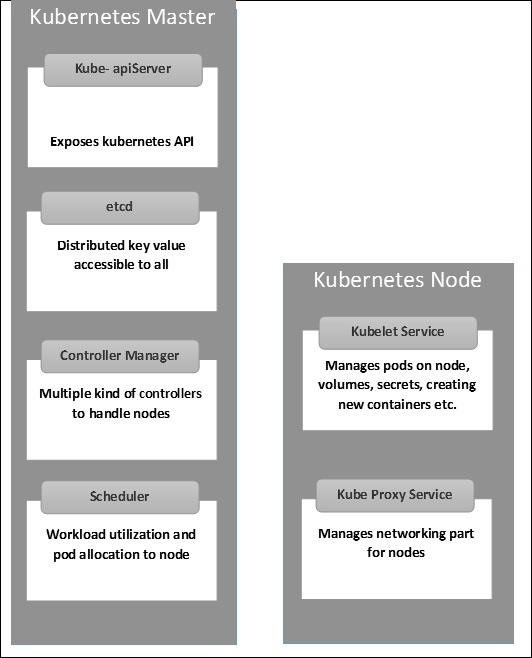


Image : Kubernetes - Master and Node Structure

Reference: <https://www.tutorialspoint.com/kubernetes/kubernetes_architecture.htm>

## Services in cloud supporting container applications

### IBM Cloud Kubernetes Service

With IBM Cloud Kubernetes Service, you can create your own Kubernetes cluster to deploy and manage containerized apps on IBM Cloud. Your containerized apps are hosted on IBM Cloud infrastructure compute hosts that are called worker nodes. You can choose to provision your compute hosts as virtual machines with shared or dedicated resources, or as bare metal machines that can be optimized for GPU and software-defined storage (SDS) usage. Your worker nodes are controlled by a highly available Kubernetes master that is configured, monitored, and managed by IBM. You can use the IBM Cloud Kubernetes Service API or CLI to work with your cluster infrastructure resources and the Kubernetes API or CLI to manage your deployments and services.

**Why should I use IBM Cloud Kubernetes Service?**

IBM Cloud Kubernetes Service is a managed Kubernetes offering that delivers powerful tools, an intuitive user experience, and built-in security for rapid delivery of apps that you can bind to cloud services that are related to IBM Watson™, AI, IoT, DevOps, security, and data analytics. As a certified Kubernetes provider, IBM Cloud Kubernetes Service provides intelligent scheduling, self-healing, horizontal scaling, service discovery and load balancing, automated rollouts and rollbacks, and secret and configuration management. The service also has advanced capabilities around simplified cluster management, container security and isolation policies, the ability to design your own cluster, and integrated operational tools for consistency in deployment.

### Red Hat Openshift service on IBM Cloud

OpenShift Container Platform is a private platform-as-a-service (PaaS) for enterprises that run OpenShift on public cloud or on-premises infrastructure. It runs on the Red Hat Enterprise Linux (RHEL) operating system and functions as a set of Docker-based application containers managed with Kubernetes orchestration.

### Amazon ECS on AWS Cloud

Amazon ECS is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. It deeply integrates with the rest of the AWS platform to provide a secure and easy-to-use solution for running container workloads in the cloud and now on your infrastructure with Amazon ECS Anywhere.

## Hybrid Application scenarios

Legacy modernization is a priority in today’s digital business environment. With rapid technological advancements, increased customer demand for digitized services, and the growing popularity of DevOps and agile approaches — rigid monolithic architectures are no longer sufficient.

What’s the solution? Companies have tried several different approaches to modernizing their existing legacy systems, but each one proved to be sub-optimal.

OpenLegacyDismantling and replacing legacy systems is time-consuming, tedious, and risky. Leaving them intact and adding layers of middleware to get the job done results in bloated code and often leads to complex maintenance issues. It’s also a huge roadblock preventing organizations from adopting modern DevOps practices.

Hybrid IT allows organizations to abstract the choice of the development environment and leverage the possibilities of microservices and cloud-native architecture to integrate their legacy systems.

Hybrid integration allows you to embrace technological innovations and deploy services in the cloud, as well as integrate the data between on-premise and cloud systems. This solves the problem of legacy modernization by allowing you to connect your existing IT infrastructure with modern systems and technologies, mobile and cloud applications, and efficiently develop and deploy digital services to meet the demands of your customers.

The concrete usage scenarios of a hybrid cloud infrastructure are as follows:

**1: Safe harbor: private environment as gateway to the cloud world**

For companies, a hybrid cloud solution can serve as a gateway to the cloud world. The private cloud environment functions as a protected space for first steps and safe testing until the desired solution runs stably on cloud servers – in a second step it can be ported promptly into the public cloud if required. What is important here, however, is a hybrid cloud infrastructure with a uniform software and hardware basis to ensure maximum compatibility between the public and private clouds.

A suitable starting point could be legacy applications, for example: T-Systems' transformation projects show that around two thirds of business applications can basically be transferred to the cloud. This requires a thorough analysis and inventory of the existing IT landscape. T-Systems, for example, offers the so-called Cloudifier with standardized transformation services for entry and transition to the cloud.

**2: Burst scenario – the public cloud as a buffer for peak loads**

Whether it’s the busy Christmas period in e-commerce, data-intensive simulations in product development or Big Data analyses in research: Many companies only need high-performance computing capacities from time to time rather than permanently. This is a case for the hybrid cloud, in which private and public clouds can be smoothly combined to form what is known as cloud bursting. This enables companies to flexibly absorb peak loads at any time by switching storage and computing resources on and off as required.

However, some companies need to store and process certain data in their own data center or in a private cloud. For example, to ensure they are protected against industrial espionage, to adhere to compliance regulations or to benefit from the lowest possible latencies. But the public cloud can also be an overflow basin for such companies: "It depends on how such scenarios are implemented in detail. If business-critical data always remains in the private environment and only the compute service comes from the public cloud, cloud bursting can also be feasible for sensitive company data," says Sascha Smets, Senior Product Manager at T-Systems. “This can be achieved either by anonymizing the data that is to be processed in the public cloud. Or by only providing applications in the public cloud with the exact information they need for a specific computing process. This makes the interaction secure and also saves bandwidth."

**3: Backup and disaster recovery with the hybrid cloud**

Hybrid cloud scenarios can also be used as backup or disaster recovery solutions. There are several possibilities for this kind of implementation – depending on the requirements of the respective company. On the one hand, the public cloud is an inexpensive long-term storage solution. For maximum security, for example, data can be stored in encrypted form in the Object Based Storage.

Companies that do not want to store certain data in the public cloud but still want to store it redundantly can also set up two private, separate availability zones (AZs) in which they mirror their systems. The distance between the AZs is important. They should be far enough apart so that both AZs don’t fail simultaneously in the event of a possible disaster such as a flood or fire. But close enough to benefit from the lowest possible latencies. "A benchmark that has become established among companies is a distance between the data centers of around 20 to 30 kilometers," says Sascha Smets.

**4: Application development with DevOps – develop once, run anywhere**

Modern, agile application development increasingly integrates development, testing and operation. While the respective teams used to work independently and separately from each other, today they are linked by cooperation in a DevOps model. Hybrid cloud platforms enable teams to develop software faster and to shorten release procedures. Applications can be ported as needed between teams and their respective private or public environments.

What is important here is a hybrid cloud infrastructure that ensures a seamless transition between the public and private environments. The Open Telekom Cloud Hybrid, for example, offers DevOps teams this kind of unified environment, which is based on the same hardware and software for public and private clouds. Each developed application can run in both the public and private environments – in line with the motto: develop once, run anywhere. And this is true even if the company in question doesn’t work with container technology.

**5: Real-time data processing – minimum latency, maximum power**

Edge Computing and the hybrid cloud cultivate a close relationship, because even with Edge Computing, companies can make use of decentralized computing and storage capacities as required. However, these are not located in a remote data center, but close to the action, at the edge of a network – hence the term "Edge Computing." Why is this necessary? One example is the real-time processing of data, where latencies can be kept as low as possible. For example, the transmission and processing of sensor data for autonomous driving practically require a data center at every intersection. The situation is similar for the control of industrial robots with AI algorithms, because low latencies also play a central role here.

"There is a growing demand for IT resources that can handle processes with virtually no latency," says Sascha Smets. "We will soon meet this demand with our new Edge Cloud offering: mini data centers for real-time applications based on the Open Telecom Cloud technology, which we can install and operate directly on our customers' premises if required.”

**6: IT departments as service providers and cloud brokers**

The so-called shadow IT is flourishing – to the chagrin of many companies. According to Forrester Research, almost half of employees in companies now use technologies without the knowledge of their IT departments. Hybrid cloud solutions offer IT departments the opportunity to change things by building a unified service catalog and establishing themselves as the company’s cloud brokers.

If the private and public parts are based on the same technology, the IT department will find it easier to build this type of unified service catalog. For applications with sensitive data that have to stay in the company, IT can then offer virtual machines with appropriate specifications. For less critical workloads that can operate in a public cloud, IT offers a VM with exactly the same specification – but at significantly lower cost. This means that users in specialist departments can select the services they need from private or public instances at the click of a mouse, without neglecting requirements such as scalability, security and governance.

**7: Provide remote locations without network infrastructure with the cloud**

Whether it be an oil rig, space station or remote research facility: Not every location can be easily supplied with a fast Internet connection. With the Open Telekom Cloud Hybrid, companies can also use IT resources in remote locations as a private cloud – regardless of the network connection. Telekom can implement the necessary servers directly where they are needed. A connection to the Internet or the public cloud is not absolutely necessary for operating them.

**8: Outlook: Using thermal energy from data centers sensibly**

Servers give off a lot of heat: To ensure an optimum operation, the hardware in data centers is usually cooled. This results in double the energy requirements – once for server operation and then again for cooling. But it’s also possible to make sensible use of the thermal energy from servers. For example, supplying buildings with hot water and heating energy. In this way, companies not only save on the electricity for cooling the server cabinets, but also heating costs and they use their resources in a worthy and sustainable way. If the number of hybrid cloud architectures in companies doubles by 2021, more and more data centers will emerge in the future that are suitable for decentralized heat supply.

## Hybrid architectures and best practices

**Four Architecture Choices for Application Development in the Digital Age**

Increasingly, businesses are going through a digital transformation journey to meet evolving consumer needs. Customers are also more and more likely to be using social networks, mobile applications, and digital technologies. Due to this change, digital strategy is now an integral part of the overall business strategy.

Many enterprises are obtaining computing power through cloud services platforms via the internet and adopting a cloud-first strategy for most application development. This has furthered a change in application design—previously, functionality and statefulness were prioritized, but now most consumer-facing applications are moving to Software-as-a-Service (SaaS) and digital platforms. The application design focus is now much more focused on user experience, statelessness, and agility.

Choosing the right application architecture depends on your business requirements. In this post, we will examine four architecture choices for enabling digital transformation, depending on general business needs.

**Traditional 3-tier application architecture**

We all know about the 3-tier application architecture—it is a client-server architecture with a typical structure consisting of the presentation layer, application layer, and database layer.

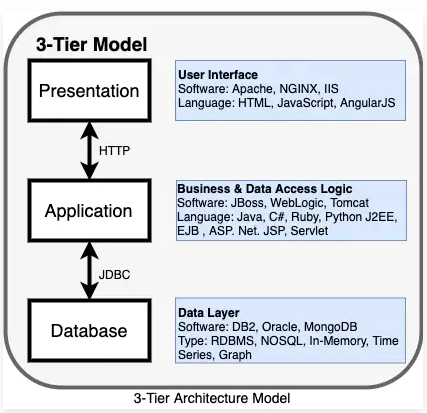


Image : 3 tier Architecture model

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

It has a user interface, business/data access logic, and data access. Many enterprise applications were created using the simple 3-tier application architecture.

**What is the issue with 3-tier application architecture?**

Simply speaking, the 3-tier application model is outdated. It was designed for application development before the proliferation of public cloud and mobile applications and has had difficulty adapting to the cloud.

Over time, an application can become too large and complex to make frequent changes. Not only that, but it also requires the maintenance of at least three layers of hardware and software, which can be inefficient for the business.

The 3-tier application model is also frequently called a monolithic architecture. These days, we have multiple new architecture models, and below, we will examine a few that are available now in the cloud era.

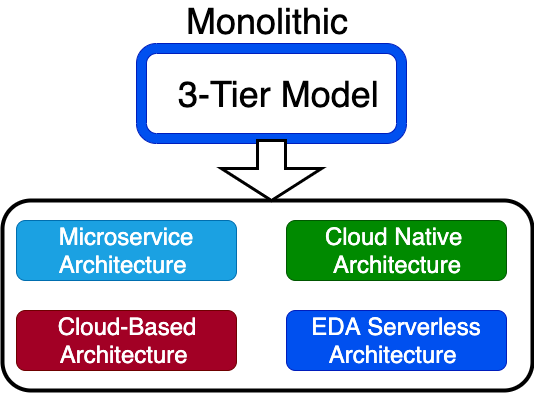


Image : Monolithic 3 tier model

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

1. **Microservices architecture**

In a cloud model, complex applications designed as a collection of services and data are fully decoupled from the application. Microservices are an architectural style that structures the application as a collection of services. Each service can be written in a different programming language and tested separately. They are independently deployable and organized around business capabilities.

Take the example of an e-commerce application developed using microservices architecture. Each microservice can focus on a single business capability (e.g., shopping cart, search, customer review). Each of these can be a separate service written in different programming languages, deployed in different infrastructure, and managed by different teams.

Each service communicates with the others using a lightweight protocol. For a 3-tier, we all know about the Model View Controller (MVC) framework. Sidecar, Ambassador, and Adapter are some of the frameworks that support microservices architectures.

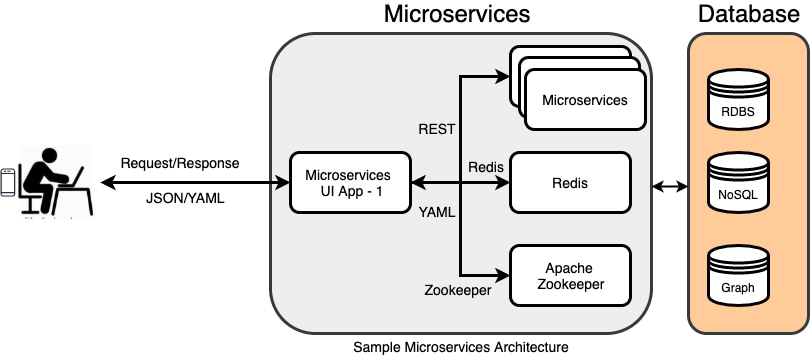


Image : Sample microservices architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

**Microservices architecture vs. monolithic architecture**

In monolithic architecture, all these components coexist as a single module managed (mostly) by a single team—everything is bundled together. If you need to update, you need to deploy the entire application, and this slows down changes for larger complex applications. For smaller applications, monolithic architecture is often the best solution.

**Microservices, containers, and Kubernetes**

One of the best choices for creating and running microservices application architectures is by using containers. Containers encapsulate a lightweight virtualization runtime environment for your application and allow you to move the application from the developer's desktop all the way to production deployment. You can run containers on virtual machines or physical machines in the majority of available operating systems. Containers present a consistent software environment, and you can encapsulate all dependencies of your application as a deployable unit. Containers can run on a laptop, bare metal server, or in a public cloud.

Many organizations use Kubernetes to manage containers and ensure that there is no downtime. Kubernetes provides container orchestration in multiple hosts and is used for container lifecycle management. You can automate deployment, auto-scale your application, and build fast and ship fast using Kubernetes.

Red Hat OpenShift is one of the most popular leading hybrid cloud enterprise container platforms. Many public cloud providers offer Containers-as-a-Service (CaaS). Some of the other Kubernetes engines available are IBM Cloud Kubernetes Service, open source Kubernetes, AWS (EKS, ECS, and Fargate), Google GKS, and Azure AKS.

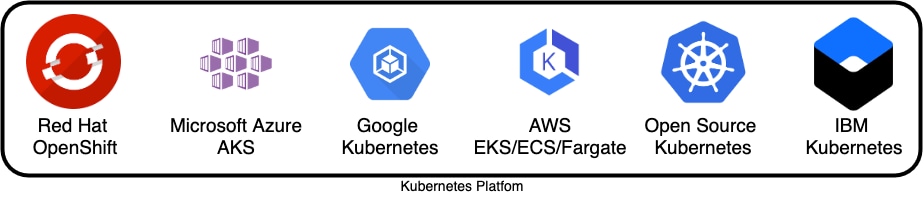


Image : Kubernetes platform

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

Usually each microservice is built by a different small team, and they choose their programming language and deployment schedule. A service mesh like Istio is used by enterprises to govern communication between microservices and management. In a service mesh, requests are routed through proxies (such as Sidecar) between microservices.

2. **Cloud native architecture**

Cloud native architecture is designed specifically for applications planning to deploy in the cloud, and microservices are a critical part.

Cloud native is an approach to building and running applications that exploits the advantages of the cloud computing delivery model. Cloud native is a term used to describe container-based environments, and it is about how applications are created and deployed, not where.

Cloud native technologies empower us to run applications in public, private, and hybrid clouds. Cloud native development is essential to getting applications to market quickly; it helps people, processes, and technologies to build, deploy, and manage apps that are ready for the cloud.

The cloud native architecture model uses DevOps, continuous integration (CI), continuous delivery (CD), microservices, and containers. Most of the enterprises use the twelve-factor methodology for designing scalable and robust cloud native applications.

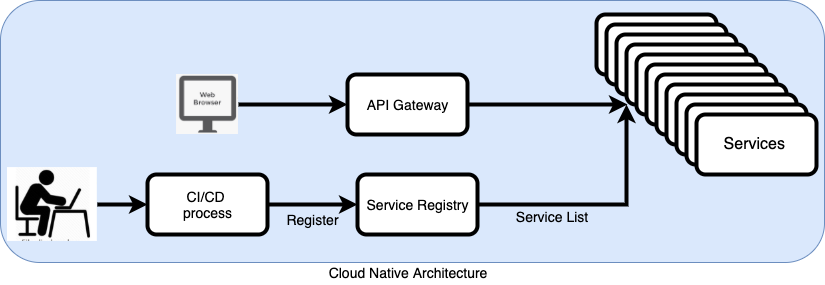


Image : Cloud Native architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

In the cloud, applications must be able to run concurrently in multiple nodes, share a configuration/session state, have a centralized logging mechanism, and be able to deploy using DevOps and a CI/CD process. Many cloud providers give guidelines for cloud native development—Amazon Web Services (AWS) has its well-architected framework, Google has various guides on how to build cloud native applications, and Microsoft Azure has its cloud patterns guide.

Usually, cloud native applications are stateless by nature. The services communicate with each other using REST-based protocols or messaging. The API Gateway, container registry, message-oriented middleware (MOM: Publish/Subscribe or Request/Response), service mesh, and orchestrations could be part of cloud native architecture.

3. **Event-driven serverless architecture**

Event-driven architecture (EDA) is based on decoupled systems that run in response to events. An event-driven architecture uses events to trigger and communicate between decoupled services. EDA has been here for a long time, but it now has more relevance in the cloud.

So, what is new? If properly used, it can provide a significant increase in agility, cost savings, and operational benefits. The distributed serverless EDA can execute code known as functions that scale automatically in response to a REST API or an event trigger.

For the serverless model, there is no server management needed. The serverless model is also quickly scalable (so quick updates and deployment are possible) and it is stateless.

Here are some of the currently available cloud serverless services from different cloud providers:

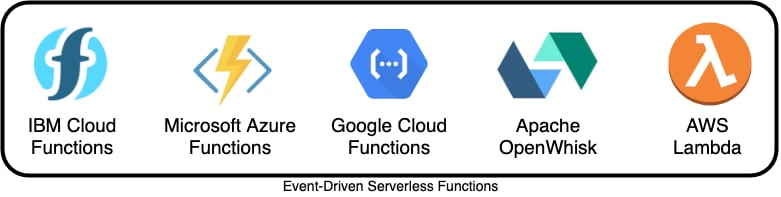


Image : Event-Driven Serverless Functions

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

**Types of serverless**

Functions-as-a-Service (FaaS): Upload pieces of functionality to the cloud and let these pieces be executed independently.

Backend-as-a-Service (BaaS): Utilize services from a third party, such as application management, database management, and cloud storage.

Mobile-Backend-as-a-Service (MBaaS): Functions for mobile applications.

4. **Cloud-based architecture**

How can we make monolithic applications work well in a cloud environment? Cloud-based architecture is best suited for building a modern web application (static/dynamic websites), deploying a web application, connecting to a database, and analyzing user behavior.

A traditional cloud-based application architecture involves load balancers, web servers, application servers, and databases. It can benefit from cloud features such as resource elasticity, software-defined networking, auto-provisioning, high availability, and scalability.

This type of architecture is ideal for organizations that don't have to worry about maintaining a server. The serverless functions support different programming languages, such as PHP, Java, .NET, Node.js, Python, Ruby, Docker, and Go.

API Gateway is an important service that makes it easy for developers to create and publish secure APIs. The APIs will act as a front door for applications to access data and business logic. It also takes care of authorization and access control. Developers use API Gateway to invoke different serverless functions for different API calls.

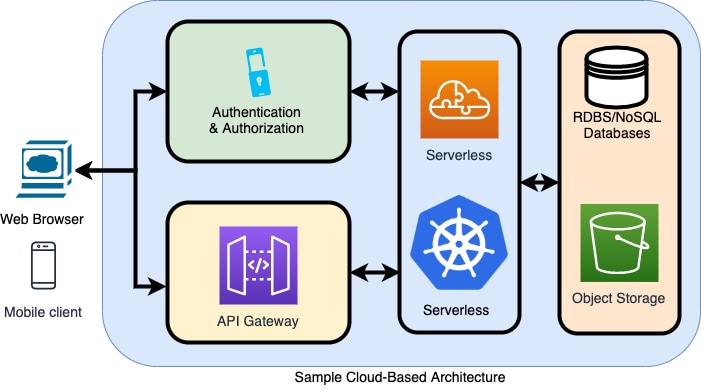


Image : Sample Cloud Based architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

**How to decide which architecture model is best for your application?**

The decision you make when choosing an architecture model can influence the success or failure of your project. You should make your choice based on your application and on non-functional requirements.

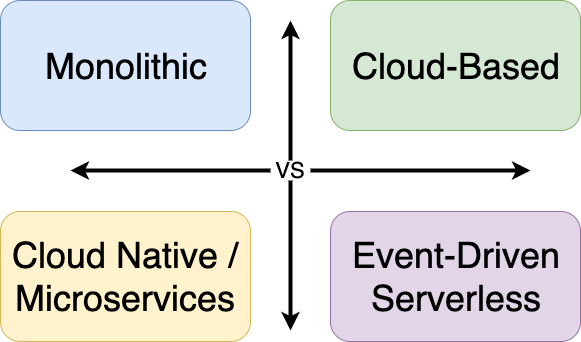


Image : Sample Cloud Based architecture

Reference: <https://www.ibm.com/cloud/blog/four-architecture-choices-for-application-development>

Consider the following before choosing an architecture for your app project:

* Is it monolithic-first or microservice-first? (For smaller projects with a simple application requirement, monolithic may be a right choice.)
* Is your team ready to utilize microservices?
* Does your team have an existing cloud-based DevOps and CI/CD process?
* What is your hosting model? Private, public, hybrid?
* How does the application architecture affect your project?
* Does a combination of multiple architecture model work for you?
* Do you need persistence and sessions for your applications?

References

1. <https://cloudacademy.com/blog/cloud-migration-benefits-risks/>
2. <https://www.ibm.com/cloud/learn/application-migration#toc-applicatio-UwMDGjMF>
3. <https://cloud.netapp.com/blog/cloud-migration-strategy-challenges-and-steps>
4. https://aws.amazon.com/devops/
5. <https://www.ibm.com/cloud/devops>
6. <https://cloud.ibm.com/docs/apps?topic=apps-devops-toolchains>
7. <https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-toolchains_getting_started&interface=ui>
8. <https://www.ibm.com/cloud/architecture/toolchains/>
9. <https://www.ibm.com/cloud/architecture/tutorials/introduce-develop-cloud-foundry-app-toolchain/>
10. <https://cloud.ibm.com/docs/ContinuousDelivery?topic=ContinuousDelivery-devops_intro>
11. <https://newrelic.com/blog/best-practices/what-is-serverless-architecture>
12. <https://www.baeldung.com/cs/serverless-architecture>
13. <https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless>
14. <https://www.datadoghq.com/knowledge-center/serverless-architecture/>
15. <https://www.ibm.com/cloud/learn/containers>
16. <https://www.tutorialspoint.com/docker/>
17. <https://cloud.netapp.com/blog/cvo-blg-understanding-red-hat-openshift-container-platform>
18. <https://www.openlegacy.com/blog/hybrid-cloud>