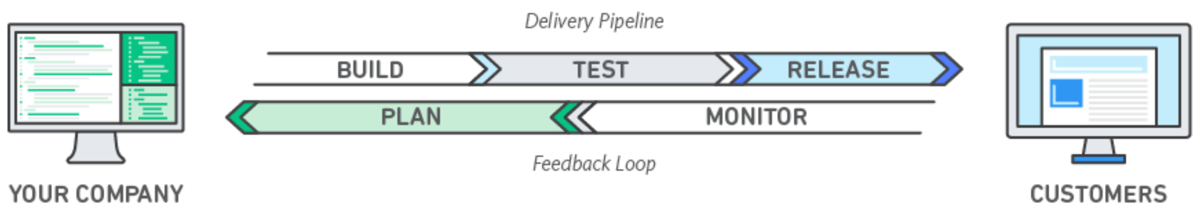
1. What is DevOps

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



2. 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.

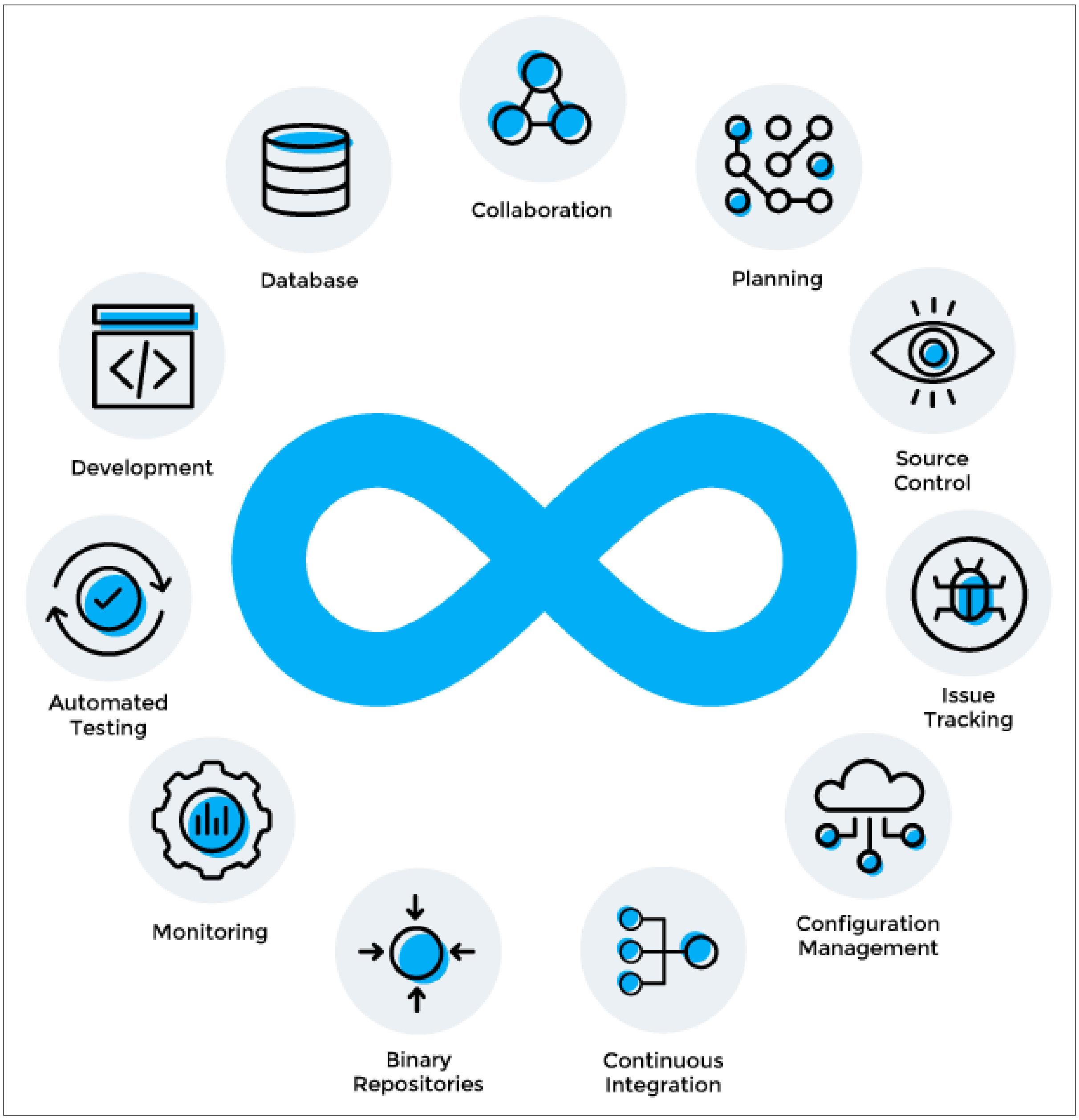
These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning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

3. What Is a DevOps Toolchain?

Key DevOps fundamentals revolve around the concepts of continuous integration, continuous delivery, automation, and collaboration. Since DevOps is more of a practice than technology,

there's no single tool that can do justice to all stages of software development. Rather, DevOps forms a series of tools.
There are a number of open-source DevOps tools available. Clubbing them together based on your needs makes a **DevOps toolchain**. This makes product delivery faster and more efficient. A toolchain is basically a set of various tools that solves a particular problem.

As mentioned above, different tools are used at different stages of the software development cycle.



Collaboration

The greatest catch of the DevOps culture is collaboration and communication between different teams. Different teams like development, testing, and product coordinate and work to automate this entire process. Collaboration tools help teams work together regardless of time zones and locations. Faster communication means faster software releases. A few examples of collaboration tools are Slack, Campfire, and Skype.

Planning

Stakeholders, clients, and employees working with different teams should have common goals. Therefore, transparency among all participants is important. Planning tools provide this transparency. A couple of examples of planning tools are Asana and Clarizen.

Source Control

You need a centralized storage location for all your data, documentation, code, configurations, files, etc. Data from this source control can then further be divided into different branches for teams to work on. Source control tools give you these features to exploit. A few examples of source control tools are Git, Subversion, and SVN.

Issue Tracking

An increase in transparency results in clearer vision, making it easier and faster to track issues. There are issue tracking tools, but there is a condition: all the teams should be using the same tracking tool. A few examples of these issue tracking tools are Jira, ZenDesk, and Backlog.

Configuration Management

Wouldn’t it be perfect if all your system was automatically configured and updated without you having to worry about it? Configuration management tools are meant for that. These tools help manage your infrastructure as code, which then avoids configuration drifts across environments. A few examples of configuration management tools are Ansible, Puppet, and Chef.

Continuous Integration

A good software development cycle gets the code developed in chunks by different teams and then continuously integrates them. The codes might work perfectly fine individually but can create issues when integrated. Continuous integration tools let you detect errors quickly and resolve them faster. A few examples of continuous integration tools are Bamboo, Jenkins, and TeamCity.

Binary Repositories

A product might be getting developed on a daily basis or an hourly basis. The code needs to be flowing smoothly from the developer’s machine to the production environment, thus a repository manager is a good way to bridge this gap. Repositories contain collections of binary software artifacts, metadata, and code. A few examples of binary repositories are Artifactory, Nexus, and Maven.

Monitoring

As the name suggests, monitoring is a must in DevOps for smooth execution. Monitoring tools ensure service uptime and optimal performance. A couple of examples of monitoring tools are BigPanda and Sensu.

Automated Testing

The entire integrated code needs to be tested before passing it to the build. The quicker the feedback loop runs, the quicker you reach your goal. A few examples of automated testing tools are Telerik, QTP, and TestComplete.

Development

Another great concept of DevOps that allows the application deployment to be frequent and reliable is development. Deployment tools let you release your products faster to the market. A few examples of development tools are the Docker toolset, and IBM uDeploy.

Database

Finally, there’s handling the data. Data is valuable for getting insights, and every application development requires a lot of data. Database management tools help you handle cumbersome data with ease. Some examples of database management tools are RazorSQL, TeamDesk, etc.

Now that you know what a DevOps toolchain is, let’s also explore the need for it.

4. Why Do We Need a DevOps Toolchain?

DevOps culture brings you good results in terms of product delivery and money. Companies require developers with the skills and expertise of using different DevOps tools. Is it worth spending so much on the skilled employees and changing the entire company's infrastructure? Well let's have a look.

Faster deployments: Using these tools automates most of the stages of the software development cycle. Agile and rapid product deliveries are the result of using standardized pipelines. Consequently, businesses that innovate faster win the competition.

Fine-tuned incident control: Humans are careless and make reckless mistakes—hence, it's better to trust tools. Using a standardized pipeline and infrastructure makes various teams respond faster and more effectively during an incident.

Quality assurance: Resolving software defects quickly and certainly with precision is pretty difficult. But DevOps tools make it seem like a walk in the park. The DevOps toolchain brings out the best product with the best quality, as quality is one of the major selling points for most of the products.

5. How Do We Create a DevOps Toolchain?

There are five main aspects of creating a DevOps toolchain.

Acceptance: The first step to making a revolutionary change is accepting that something is wrong and, furthermore, accepting that change is required. If your developments aren't moved to production quickly, then you most definitely need another toolchain. In other words, you need a toolchain that moves things faster.

Inspiration: There are many companies that have already adopted DevOps and have benefited from it. Techies are always ready to contribute. Read some of their success stories, reach out and connect with them in different tech communities, and learn from them.

Analysis: Analyze your current system, as well as the tools that you're using. Find out how much time each step takes and what the accuracy is. This will help you identify the loopholes in your current system. You now know what needs to be changed.

Build: Once you know what has to be changed, you can go ahead and start selecting the best tools for your requirements. Build the prototype of your toolchain. This is the time where you put all the theoretical knowledge into practice. Improvise your current metrics using these tools.

Strategy: Businesses these days are very dynamic. The competition demands a scale-up at some point. Hence, your toolchain should be capable of handling unexpected situations. You need to maintain, upgrade, and configure your tools over time. Plan your long-term toolchain support strategy.

6. Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

1. Docker Client

Docker client uses **commands** and **REST APIs** to communicate with the Docker Daemon (Server). When a client runs any docker command on the docker client terminal, the client terminal sends these docker commands to the Docker daemon. Docker daemon receives these commands from the docker client in the form of command and REST API's request.

Docker Client uses Command Line Interface (CLI) to run the following commands -

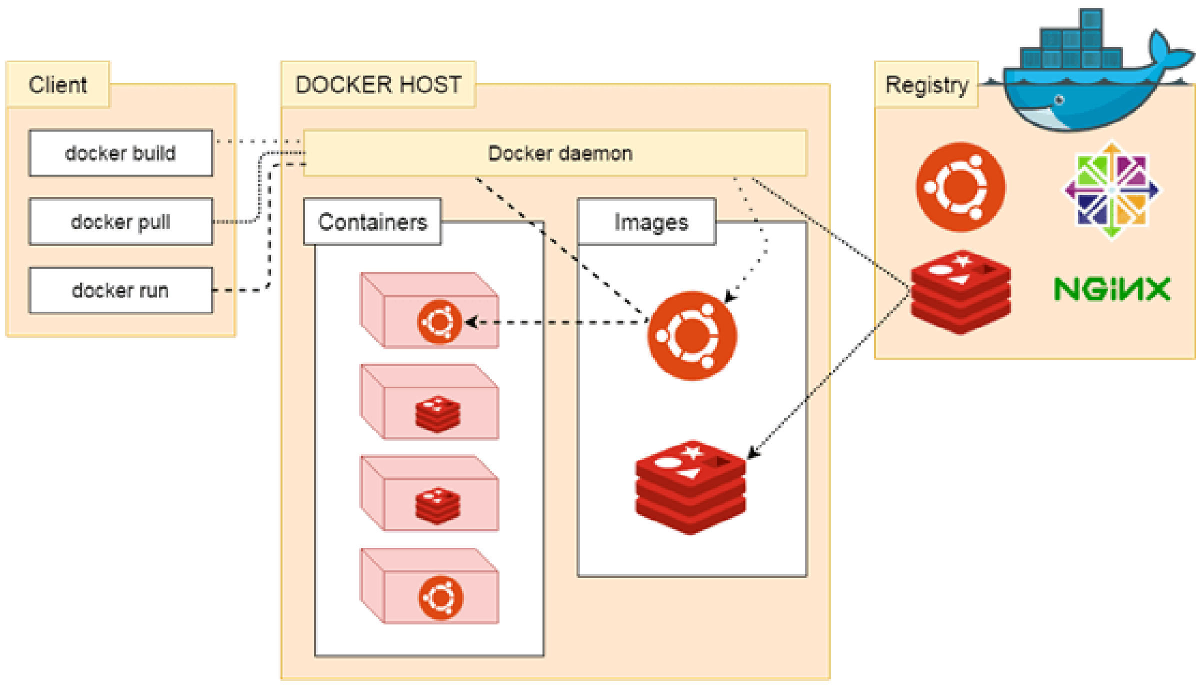
docker build

docker pull

docker run

2. Docker Host

Docker Host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks, and storage.



3. Docker Registry

Docker Registry manages and stores the Docker images.

There are two types of registries in the Docker -

Pubic Registry - Public Registry is also called as **Docker hub**.

Private Registry - It is used to share images within the enterprise.

Docker Objects

There are the following Docker Objects -

Docker Images

Docker images are the **read-only binary templates** used to create Docker Containers. It uses a private container registry to share container images within the enterprise and also uses public container registry to share container images within the whole world. Metadata is also used by docket images to describe the container's abilities.

Docker Containers

Containers are the structural units of Docker, which is used to hold the entire package that is needed to run the application. The advantage of containers is that it requires very less resources.

In other words, we can say that the image is a template, and the container is a copy of that template.

Docker Networking

Using Docker Networking, an isolated package can be communicated. Docker contains the following network drivers -

- **Bridge -** Bridge is a default network driver for the container. It is used when multiple docker communicates with the same docker host.
- **Host -** It is used when we don't need for network isolation between the container and the host.
- **None -** It disables all the networking.
- **Overlay -** Overlay offers Swarm services to communicate with each other. It enables containers to run on the different docker host.
- **Macvlan -** Macvlan is used when we want to assign MAC addresses to the containers.

Docker Storage

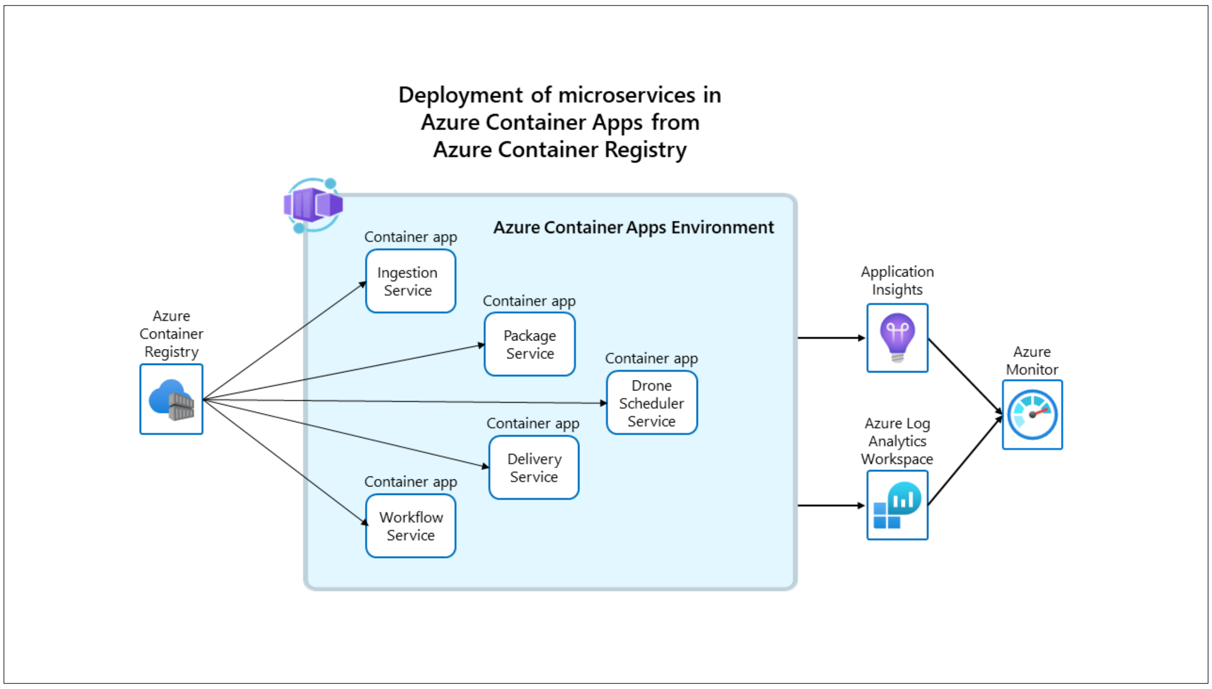
Docker Storage is used to store data on the container. Docker offers the following options for the Storage -

- **Data Volume -** Data Volume provides the ability to create persistence storage. It also allows us to name volumes, list volumes, and containers associates with the volumes.
- **Data Volume Container**: A Data Volume Container is an alternative approach wherein a dedicated container hosts a volume and to mount that volume to other containers. In this case, the volume container is independent of the application container and therefore can be shared across more than one container.
- **Directory Mounts -** It is one of the best options for docker storage. It mounts a host's directory into a container.
- **Storage Plugins -** It provides an ability to connect to external storage platforms. There are storage plugins from various companies to automate the storage provisioning process. For example
    - HPE 3PAR
    - EMC (ScaleIO, XtremIO, VMAX, Isilon)
    - NetApp

There are also plugins that support public cloud providers like:

- Azure File Storage
- Google Compute Platform.

7. DevOps for Deployment:

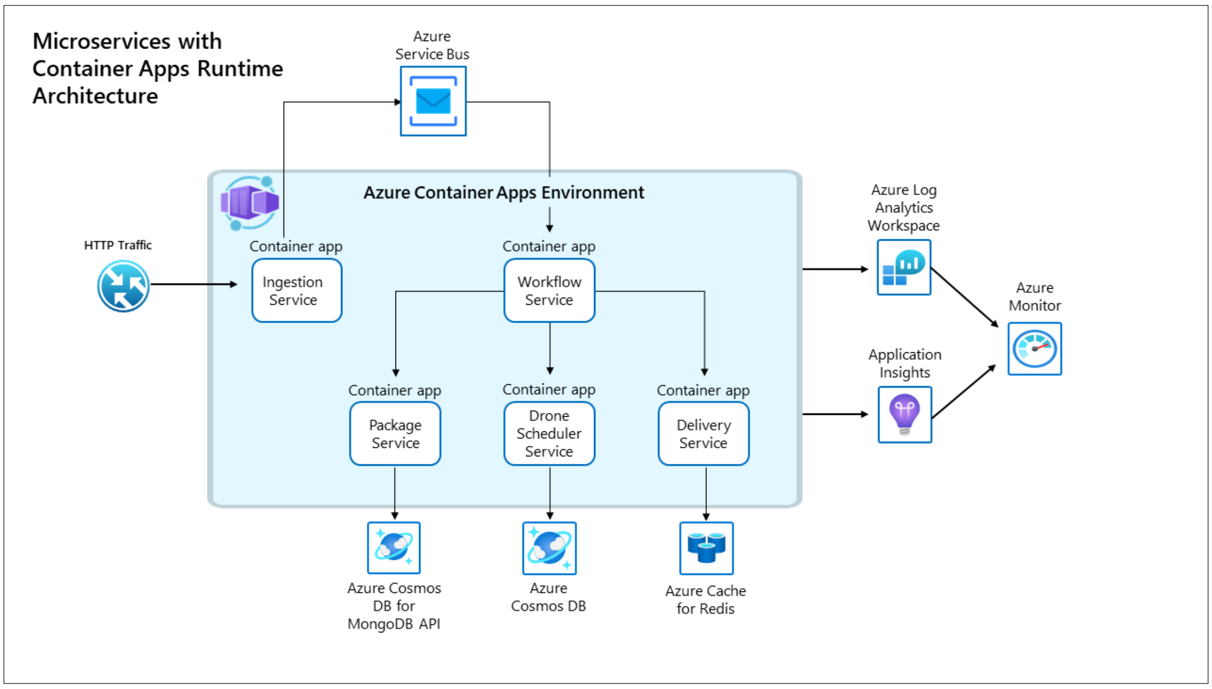


In this scenario, the container images are sourced from Azure Container Registry and deployed to a Container Apps environment.

The services sharing the same environment benefit from:

- Internal ingress and service discovery
- A single Log Analytics workspace for runtime logging

The workflow service container app is running in single revision mode. A container app running in single revision mode will have a single revision that is backed by zero-many replicas. A replica is composed of the application container and any required sidecar containers. This example isn't making use of sidecar containers, therefore each container app replica represents a single container. Since this example doesn't employ scaling, there will be only one replica running for each container app.



This diagram illustrates the runtime architecture for the solution.

Workflow

1. **Ingestion service:** Receives client requests, buffers them and sends them via Azure Service Bus to the workflow service.
2. **Workflow service:** Consumes messages from Azure Service Bus and dispatches them to underlying services.
3. **Package service:** Manages packages.
4. **Drone scheduler service:** Schedules drones and monitors drones in flight.
5. **Delivery service:** Manages deliveries that are scheduled or in-transit.

Components:

The drone delivery service uses a series of Azure services in concert with one another. Azure Container Apps is the primary component.

External storage and other components:

1. Azure Cosmos DB stores data using the open-source Azure Cosmos DB API for MongoDB. Microservices are typically stateless and write their state to external data stores. Azure Cosmos DB is a NoSQL database with open-source APIs for MongoDB and Cassandra.

2. Azure Service Bus offers reliable cloud messaging as a service and simple hybrid integration. Service Bus supports asynchronous messaging patterns that are common with microservices applications.

3. Azure Cache for Redis adds a caching layer to the application architecture to improve speed and performance for heavy traffic loads.

4. Azure Monitor collects and stores metrics and logs. Use this data to monitor the application, set up alerts and dashboards, and do root cause analysis of failures. This scenario uses a Log Analytics workspace for comprehensive monitoring of the application.

5. Application Insights provides extensible application performance management (APM) and monitoring for the services. each service is instrumented with the Application Insights SDK to monitor the app and direct the telemetry data to Azure Monitor.

6. Azure Resource Manager (ARM) Templates to configure and deploy the applications.