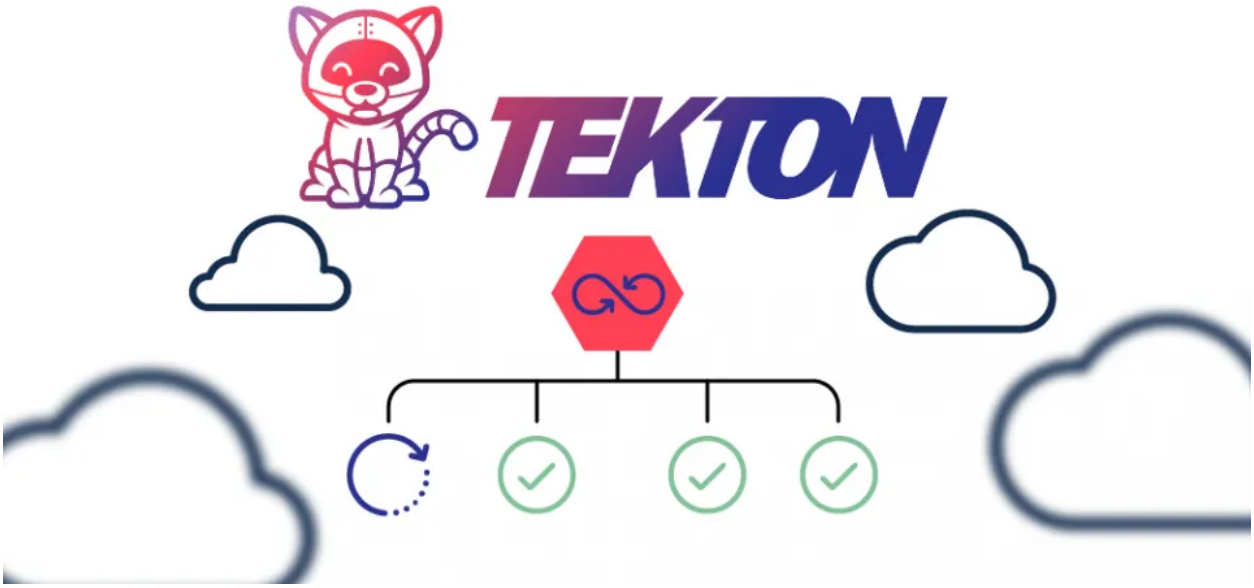
Exploring Tekton

Introduction to Tekton

What is Tekton?

Tekton is an open-source framework for creating continuous integration and continuous delivery (CI/CD) systems that are natively integrated with Kubernetes. Designed by the Continuous Delivery Foundation (CDF), Tekton allows developers to automate the deployment and management of software through a set of flexible, composable building blocks. These blocks enable users to define and run pipelines directly in Kubernetes, making it easier to manage and scale CI/CD processes using cloud-native approaches.



Evolution of Tekton as a CI/CD Tool Specific to Kubernetes

Tekton evolved from the Knative Build project, which aimed to provide build capabilities for serverless workloads on Kubernetes. Recognizing the need for a more general-purpose CI/CD solution that could harness the power of Kubernetes, Tekton was spun out as a separate project. It addressed limitations present in traditional CI/CD tools by leveraging Kubernetes' orchestration capabilities and container-based execution, allowing for greater scalability, reliability, and portability. Over time, Tekton has matured into a robust framework focusing on flexibility and extensibility, enabling teams to build complex pipelines that can be easily shared and reused.

Significance of Tekton in Cloud-native CI/CD Pipelines

Tekton's integration with Kubernetes makes it highly suitable for cloud-native environments, where applications are often built as microservices and deployed in containers. Its significance lies in several key areas:

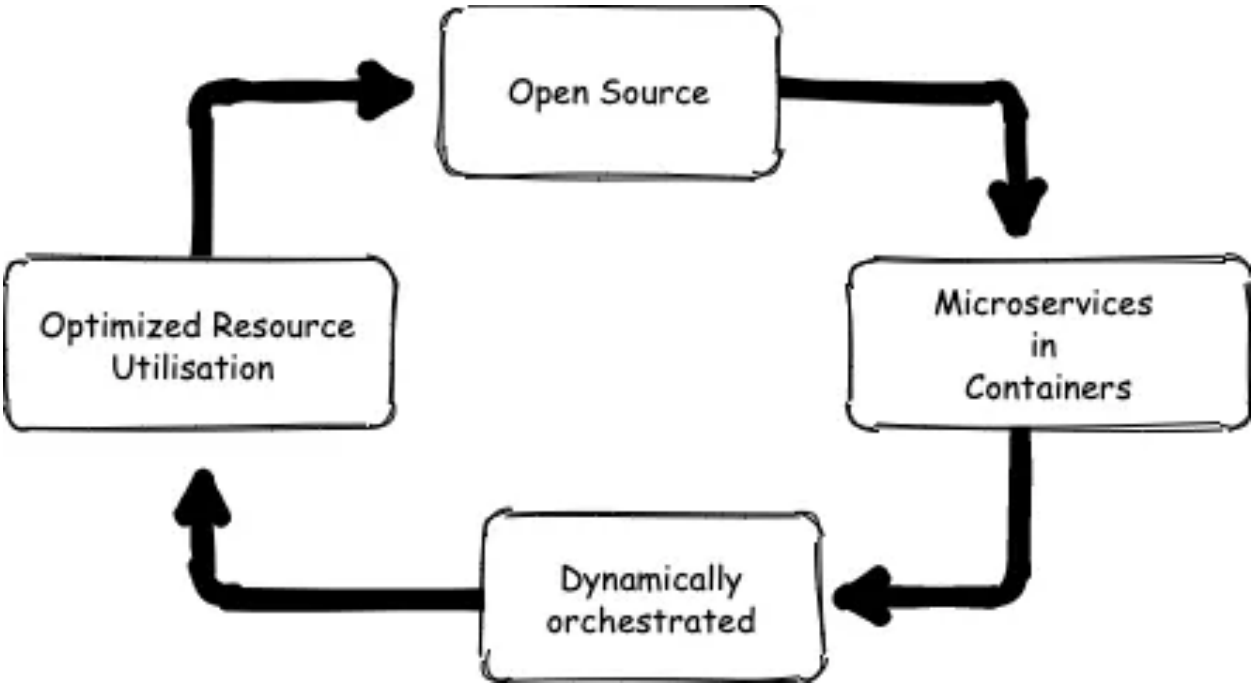
1. **Kubernetes-native Architecture**：Tekton’s components, such as Pipelines and Tasks, are defined using Kubernetes resources (CRDs), allowing seamless integration with other Kubernetes-native processes and tools.

2. **Flexibility and Extensibility**：Tekton provides reusable components that can be easily extended and customized to fit a wide range of CI/CD scenarios. Users can define complex workflows using simple YAML configurations.
3. **Containerized Workloads**：Each Task in a Tekton Pipeline is executed in its own container, which ensures consistency across different environments and simplifies dependency management.
4. **Scalability and Reliability**：By leveraging Kubernetes' capabilities, Tekton can automatically scale workloads, handle large volumes of concurrent tasks, and ensure high availability, which is crucial for enterprise-grade CI/CD systems.
5. **Community and Ecosystem**：As part of the Cloud Native Computing Foundation (CNCF), Tekton benefits from a vibrant community and a growing ecosystem of plugins and integrations, which enhances its capabilities and ensures continuous improvement and innovation.

In essence, Tekton provides a modern approach to CI/CD, aligning well with the principles of cloud-native development, making it an ideal choice for organizations looking to modernize their software delivery pipelines in a Kubernetes environment.

What is cloud-native CI/CD?

Recently we have encountered a *DevOps culture shift* with cloud-native architectures in software development. Cloud-Native is a new approach to build and run applications. *In short, we can say the most popular way to go Cloud-Native consists of Containers + Kubernetes.*



With the rise of containerization, both build and deployment stages in CI/CD pipelines continue to be containerized. It has led to the rise and development of various CI and CD tools for the Cloud-Native landscape.

Kubernetes and CI/CD

While running CI/CD pipelines in Kubernetes, some of the challenges we are going to face are:

1. Container Images instead of binaries
2. Clusters: Many environments
3. Microservices instead of monoliths (Managing dependencies between all services is going to be challenging)
4. Ephemeral environments. Below cycle happens with every step in Tekton
    a. Create container
    b. Run command
    c. Destory container

Hence, we need a solution to these challenges in the form of cloud-native CI/CD tools. Some of the tools listed in the cloud-native landscape are Spinnaker, Argo

CD, Argo Workflows, Tekton, and Jenkins X. Let's take a deep dive into Tekton now!

Core Concepts of Tekton

Overview of Tekton's Architecture: Kubernetes-native, No Central Server

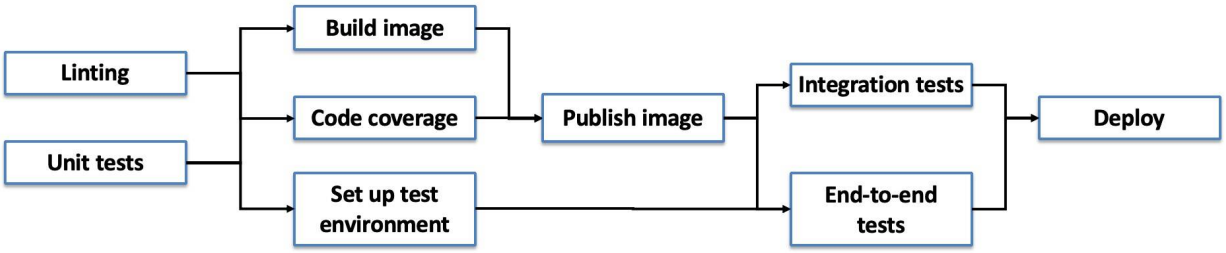
Tekton is designed as a Kubernetes-native CI/CD system that operates without a central server. This architecture leverages Kubernetes resources to manage and execute tasks, making it inherently scalable and robust.

Since there is no central server, Tekton components operate as custom Kubernetes resources, allowing them to be deployed and managed alongside other Kubernetes workloads.

This architecture removes single points of failure and enables dynamic, distributed execution of CI/CD workflows, providing enhanced reliability and integration with the Kubernetes ecosystem.

Key Components of Tekton

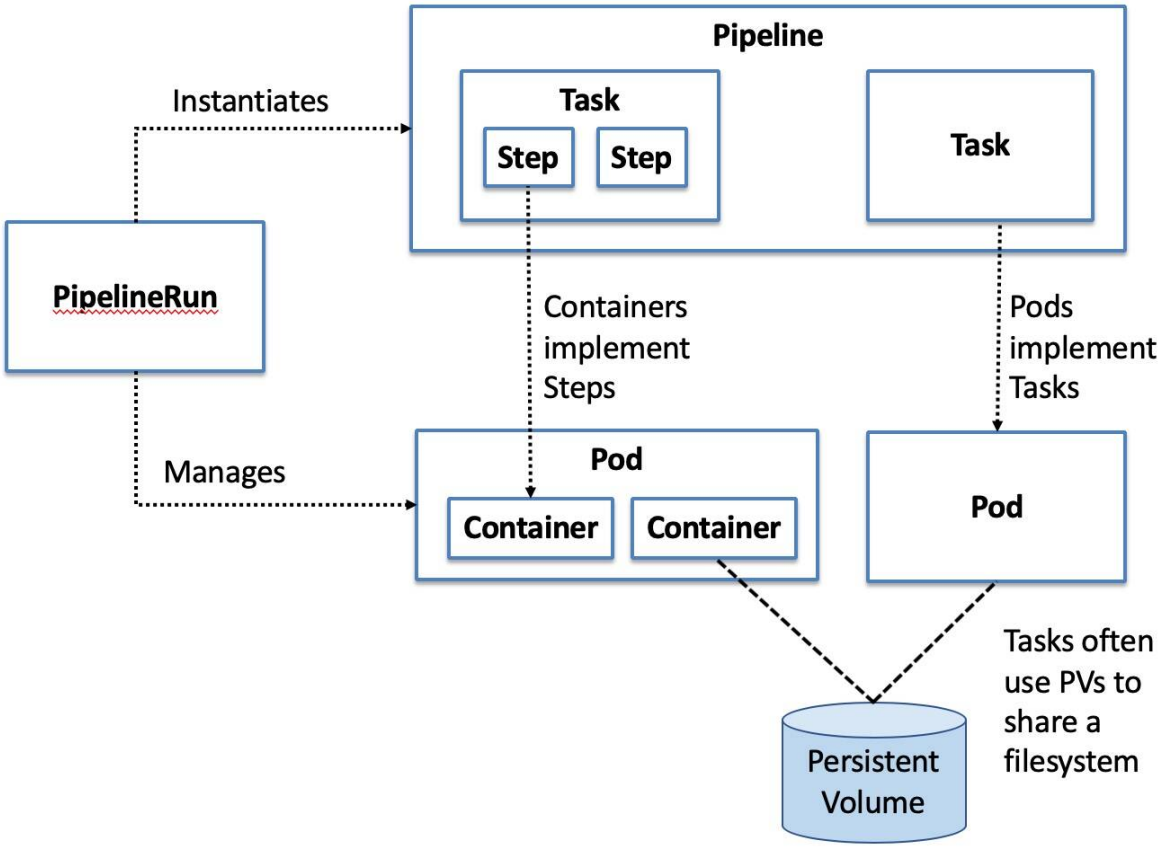
1. **Pipelines (kind: Pipeline)**
    - A Tekton Pipeline is a collection of tasks that are executed sequentially or in parallel to automate complex workflows. Each pipeline describes a sequence of steps that implement stages of a CI/CD process such as building, testing, and deploying code.



1. **Tasks (kind: Task)**

- Tasks are the primary unit of work in Tekton. Each Task is comprised of Steps, and it defines a specific part of the build or deployment process. Tasks are executed in a container, making them portable and easy to manage.
- kind: TaskRun

2. **Steps**
    - Steps are individual actions within a Task, implemented as containerized commands. Each Step runs in the context of its Task's execution environment, and multiple Steps can be defined to accomplish complex operations within a single Task.
3. **PipelineResources**
    - PipelineResources define the inputs and outputs for a Task or Pipeline. They can represent external resources like Git repositories, container images, or other digital assets that need to be accessed or produced during CI/CD execution.
    - kind: PipelineRun



4. **Workspaces**
    - Workspaces provide shared storage that Tasks can use to share data with each other. For instance, a source code workspace might be populated by a Task that checks out code from a repository, which subsequent Tasks then access to build and test the code.

Understanding Tekton's Extensibility with Custom Tasks and Integrations

Tekton is designed to be highly extensible, allowing organizations to define custom Tasks that cater to their specific CI/CD requirements. This extensibility is achieved in several ways:

- **Custom Tasks** : Users can create custom Tasks by defining new containerized steps that can perform any required operation, from running a script to making API requests. Custom Tasks can be packaged as reusable components, encouraging sharing and standardization across teams.

- **Integration with Other Tools** : Tekton can be integrated easily with other tools and platforms in the CI/CD ecosystem. For example, Tekton can trigger execution based on events from systems like GitHub, or it can integrate with monitoring tools to enhance pipeline observability.
- **Community and Ecosystem Contributions** : As an open-source project, Tekton benefits from a vibrant community that frequently contributes integrations and plugins. These contributions help extend Tekton's capabilities, providing users with a wide array of pre-built solutions for common CI/CD challenges.

The combination of Kubernetes-native architecture and extensive extensibility makes Tekton a flexible and powerful tool for building modern, cloud-native CI/CD pipelines.

Setting Up Tekton

Installation Methods: Using Kubernetes and Tekton Installation YAMLs

To set up Tekton on a Kubernetes cluster, you typically follow these steps:

1. **Prerequisites** :
    - Ensure you have a running Kubernetes cluster and sufficient permissions to install custom resources.
    - Have `kubectl` or a compatible Kubernetes client installed for interacting with your cluster.
2. **Using Installation YAMLs** :
    - Tekton installation is accomplished through the use of pre-configured YAML files. These YAML files describe the Kubernetes Custom Resource Definitions (CRDs) and other necessary components for Tekton.
    - You can download the official installation YAML from the Tekton GitHub repository or their official documentation website.
3. **Applying the YAMLs** :

- Execute the `kubectl apply` command to install Tekton using the provided YAML. This command will create several Kubernetes resources, including Tasks, Pipelines, and any required controllers or webhooks.

By applying these YAMLs, Tekton will be installed and ready to manage CI/CD pipelines in your Kubernetes environment.

Configuring Tekton in a Kubernetes Cluster

After installation, several configurations might be necessary to tailor Tekton to fit the needs of your CI/CD workflows:

1. **Custom Resource Configurations** :
    - Customize built-in Tasks and Pipelines or define new ones to match your specific CI/CD processes.
2. **Service Accounts and Permissions** :
    - Set up service accounts that have the necessary permissions to perform operations such as accessing external resources or deploying applications within your cluster.
3. **Networking and Storage** :
    - Ensure that networking configurations allow communication between Tekton components and any external systems they need to interact with.
    - Configure storage resources (e.g., Persistent Volumes) if tasks require shared storage via Workspaces.
4. **Monitoring and Logging** :
    - Enable telemetry and logging, if desired, to facilitate the monitoring of pipeline execution and troubleshooting.

Overview of Tekton Dashboard for Visualization and Management

The Tekton Dashboard provides a user-friendly interface for managing and visualizing Tekton resources and workflows:

1. **Visualize Pipelines and Tasks** :

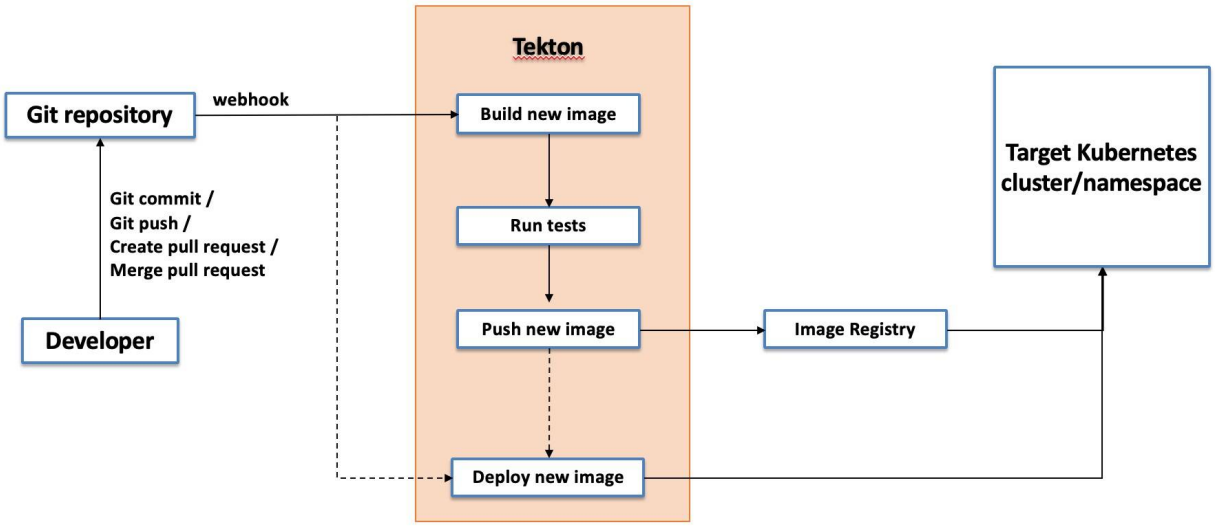
- The Dashboard displays an intuitive view of your Pipelines, allowing you to see the status of each Task and visualize the overall workflow execution.

2. **Monitor Pipeline Runs** :
   - You can monitor the real-time progress of PipelineRuns and TaskRuns, check logs, and identify any failed tasks requiring attention.
3. **Resource Management** :
   - The Dashboard allows you to manage Tekton resources directly, including creating, editing, and deleting Pipelines and Tasks without needing to use `kubectl`.
4. **User Interface Details** :
   - Access the Dashboard by deploying its components using an additional installation YAML. Once deployed, you can access it via a Kubernetes Service, usually exposed through a LoadBalancer or NodePort, depending on your cluster's configuration.

Overall, the Tekton Dashboard is an essential tool for users who prefer a graphical interface for managing CI/CD workflows, providing insights and management capabilities beyond what command-line tools offer.

Building a DevOps pipeline

Tekton gives us the pieces that we need to build a useful DevOps pipeline. The following figure shows the high level flows involved in a very simple `Pipeline` without all the linting, testing, and image scanning involved in a real world implementation.



In this example, `Pipelines` are initiated as a result of Git activity by a developer.

The previous figure shows a webhook initiating a Tekton `Pipeline`. Webhooks are outbound HTTP POST messages with a payload specific to the particular event provider. These payloads contain most of the information necessary to kick off a Tekton `PipelineRun`.

The flow in this figure shows the now-familiar process of building and publishing an image. The flow then goes on to show the new image being deployed to a target Kubernetes environment. This could be a dedicated namespace on the cluster where Tekton is running, or on one or more completely separate clusters. Deployment can occur immediately after build. For example, to a shared development or test environment, or it may occur separately as a result of Git commits to a second repository governing the target deployment environment.

ArgoCD Approach

You: Hey ArgoCD, watch this repository https://githib.com/xuyz and this folder /path/to/manifests for any change.

If change happens, then apply it to the cluster. This is URL for my repo

ArgoCD: Done.

Tekton Approach

You: Hey ArgoCD, watch this repository https://githib.com/xuyz and this folder /path/to/manifests for any change.

If change happens, then apply it to the cluster. This is URL for my repo

Tekton: Ok. Configure 3 steps.

In first one, clone the repo if there are changes.

In second one, check and validate yaml files.

In third step, apply it to cluster.