Edge Workload Configuration pattern

Article • 11/22/2022

The great variety of systems and devices on the shop floor can make workload configuration a difficult problem. This article provides approaches to solving it.

Context and problem

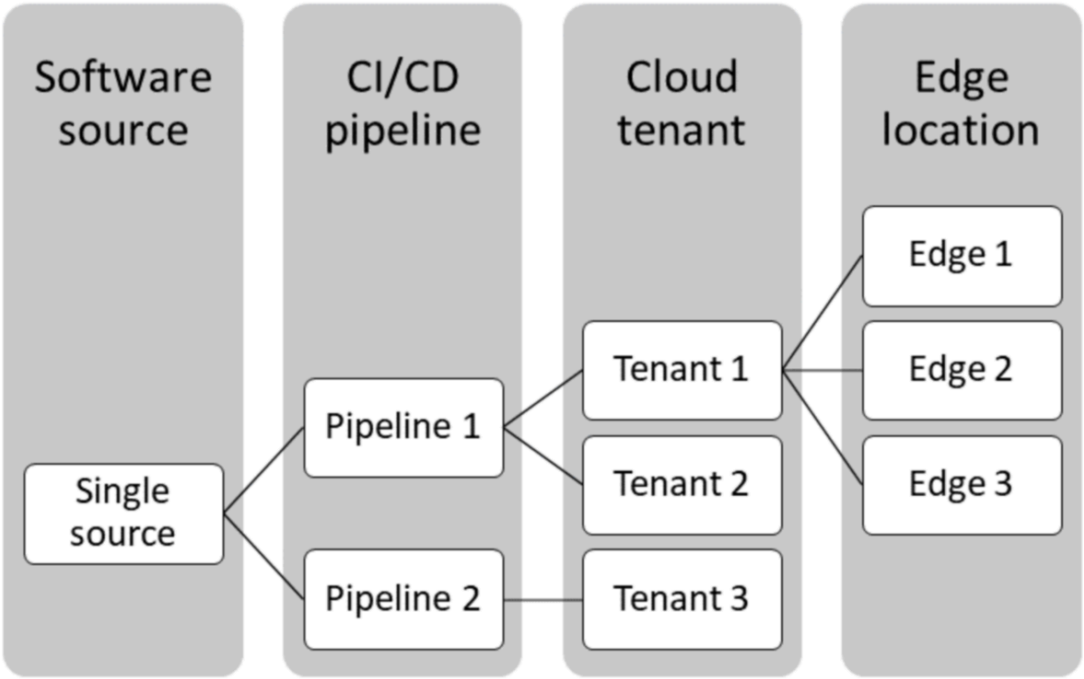
Manufacturing companies, as part of their digital transformation journey, focus increasingly on building software solutions that can be reused as shared capabilities. Due to the variety of devices and systems on the shop floor, the modular workloads are configured to support different protocols, drivers, and data formats. Sometimes even multiple instances of a workload are run with different configurations in the same edge location. For some workloads, the configurations are updated more than once a day. Therefore, configuration management is increasingly important to the scaling out of edge solutions.

Solution

There are a few common characteristics of configuration management for edge workloads:

- There are several configuration points that can be grouped into distinct layers, like software source, CI/CD pipeline, cloud tenant, and edge location:

Layers where configuration can be applied for edge workloads



- The various layers can be updated by different people.

- No matter how the configurations are updated, they need to be carefully tracked and audited.
- For business continuity, it's required that configurations can be accessed offline at the edge.
- It's also required that there's a global view of configurations that's available on the cloud.

Issues and considerations

Consider the following points when deciding how to implement this pattern:

- Allowing edits when the edge isn't connected to the cloud significantly increases the complexity of configuration management. It's possible to replicate changes to the cloud, but there are challenges with:
  - User authentication, because it relies on a cloud service like Microsoft Entra ID.
  - Conflict resolution after reconnection, if workloads receive unexpected configurations that require manual intervention.
- The edge environment can have network-related constraints if the topology complies to the ISA-95 requirements. You can overcome such restraints by selecting a technology that offers connectivity across layers, such as device hierarchies in Azure IoT Edge.
- If run-time configuration is decoupled from software releases, configuration changes need to be handled separately. To offer history and rollback features, you need to store past configurations in a datastore in the cloud.
- A fault in a configuration, like a connectivity component configured to a nonexistent end-point, can break the workload. Therefore, it's important to treat configuration changes as you treat other deployment lifecycle events in the observability solution, so that the observability dashboards can help correlate system errors to configuration changes. For more information about observability, see Cloud monitoring guide: Observability.
- Understand the roles that the cloud and edge datastores play in business continuity. If the cloud datastore is the single source of truth, then the edge workloads should be able to restore intended states by using automated processes.
- For resiliency, the edge datastore should act as an offline cache. This takes precedence over latency considerations.

When to use this pattern

Use this pattern when:

- There's a requirement to configure workloads outside of the software release cycle.
- Different people need to be able to read and update configurations.

- Configurations need to be available even if there's no connection to the cloud.

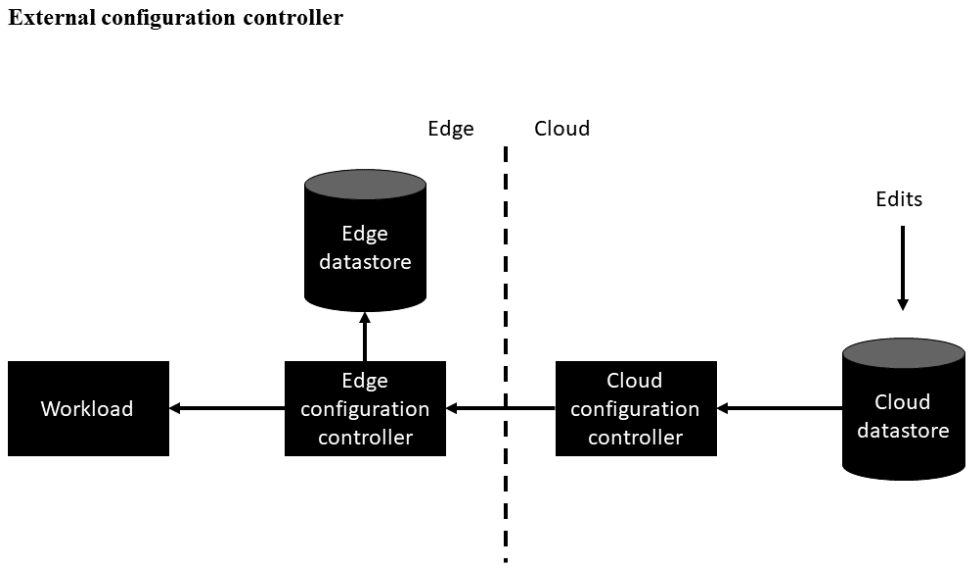
Example workloads:

- Solutions that connect to assets on the shop floor for data ingestion—OPC Publisher, for example—and command and control
- Machine learning workloads for predictive maintenance
- Machine learning workloads that inspect in real-time for quality on the manufacturing line

Examples

The solution to configure edge workloads during run-time can be based on an external configuration controller or an internal configuration provider.

External configuration controller variation



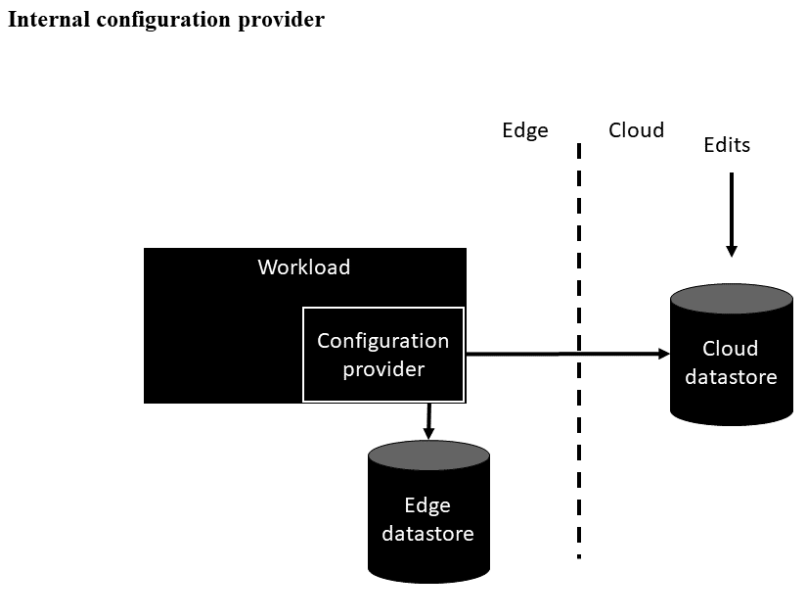
This variation has a configuration controller that's external to the workload. The role of the cloud configuration controller component is to push edits from the cloud datastore to the workload through the edge configuration controller. The edge also contains a datastore so that the system functions even when disconnected from the cloud.

With IoT Edge, the edge configuration controller can be implemented as a module, and the configurations can be applied with module twins. The module twin has a size limit; if the configuration exceeds the limit, the solution can be extended with Azure Blob Storage or by chunking larger payloads over direct methods.

The benefits of this variation are:

- The workload itself doesn't have to be aware of the configuration system. This capability is a requirement if the source code of the workload is not editable—for example, when using a module from the Azure IoT Edge Marketplace.
- It's possible to change the configuration of multiple workloads at the same time by coordinating the changes via the cloud configuration controller.
- Additional validation can be implemented as part of the push pipeline—for example, to validate existence of endpoints at the edge before pushing the configuration to the workload.

Internal configuration provider variation



In the internal configuration provider variation, the workload pulls configurations from a configuration provider. For an implementation example, see Implement a custom configuration provider in .NET. That example uses C#, but other languages can be used.

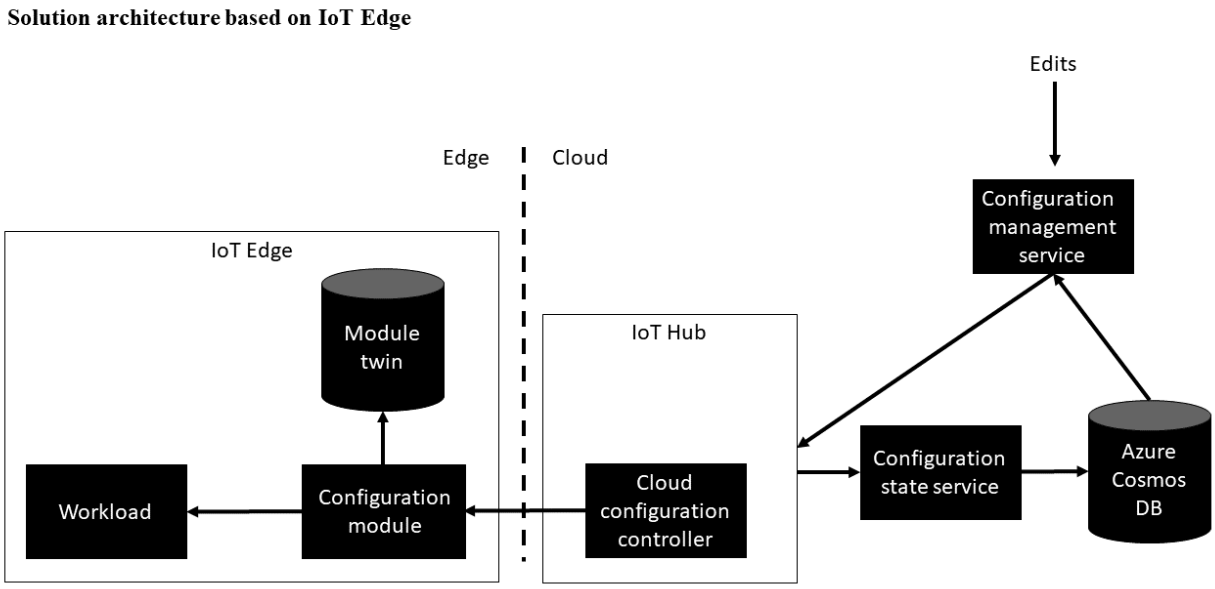
In this variation, the workloads have unique identifiers so that the same source code running in different environments can have different configurations. One way to construct an identifier is to concatenate the hierarchical relationship of the workload to a top-level grouping such as a tenant. For IoT Edge, it could be a combination of Azure resource group, IoT hub name, IoT Edge device name, and module identifier. These values together form a unique identifier that work as a key in the datastores.

Although the module version can be added to the unique identifier, it's a common requirement to persist configurations across software updates. If the version is a part of the identifier, the old version of the configuration should be migrated forward with an additional implementation.

The benefits of this variation are:

- Other than the datastores, the solution doesn't require components, reducing complexity.
- Migration logic from incompatible old versions can be handled within the workload implementation.

Solutions based on IoT Edge



The cloud component of the IoT Edge reference implementation consists of an IoT hub acting as the cloud configuration controller. The Azure IoT Hub module twin functionality propagates configuration changes and information about the currently applied configuration by using module twin desired and reported properties. The configuration management service acts as the source of the configurations. It can also be a user interface for managing configurations, a build system, and other tools used to author workload configurations.

An Azure Cosmos DB database stores all configurations. It can interact with multiple IoT hubs, and provides configuration history.

After an edge device indicates via reported properties that a configuration was applied, the configuration state service notes the event in the database instance.

When a new configuration is created in the configuration management service, it is stored in Azure Cosmos DB and the desired properties of the edge module are changed in the IoT hub where the device resides. The configuration is then transmitted by IoT Hub to the edge device. The edge module is expected to apply the configuration and report via the module twin the state of the configuration. The configuration state service then listens to twin change events, and upon detecting that a module reports a configuration state change, records the corresponding change in the Azure Cosmos DB database.

The edge component can use either the external configuration controller or internal configuration provider. In either implementation, the configuration is either transmitted in the module twin desired properties, or in case large configurations need to be transmitted, the module twin desired properties contain a URL to Azure Blob Storage or to another service that can be used to retrieve the configuration. The module then signals in the module twin reported properties whether the new configuration was applied successfully and what configuration is currently applied.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- Heather Camm | Senior Program Manager

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Azure IoT Edge
- What is Azure IoT Edge?
- Azure IoT Hub
- IoT Concepts and Azure IoT Hub
- Azure Cosmos DB
- Welcome to Azure Cosmos DB
- Azure Blob Storage
- Introduction to Azure Blob storage

Related resources

- External Configuration Store pattern

Feedback

Was this page helpful? Yes No