IoT Edge Working Group

EDGE NATIVE APPLICATION DESIGN BEHAVIORS WHITEPAPER

AUTHORS

Frank Brockners - Cisco Systems
Joel Roberts - Cisco Systems
Kate Goldenring - Fermyon
Andy Anderson - KubeStellar / IBM Research

REVIEWERS

Brandon Wick - Aarna Networks
Herve Muyal - Cisco Systems
R. Prakash - eOTF
Tomoya Fujita - Sony
Steven Wong - VMWare

CLOUD NATIVE COMPUTING FOUNDATION

PUBLISHED
October 23, 2023 (1st edition)

OBJECTIVE

Building on the **Edge Native Application Principles Whitepaper**, developed by the CNCF IoT Edge Working Group (originally published January 17, 2023), this supplemental paper includes principles that can be translated into practice by recommending design behaviors for developing applications for Edge environments.

Cloud native application design best practices have been well established, with a notable example being the "**Twelve-Factor App methodology**". Edge native application design builds on cloud native application design. However, several qualities of the Edge and the Cloud differ. Consequently, Edge native application design includes several cloud native design principles while also expanding them to meet the unique requirements of the Edge.

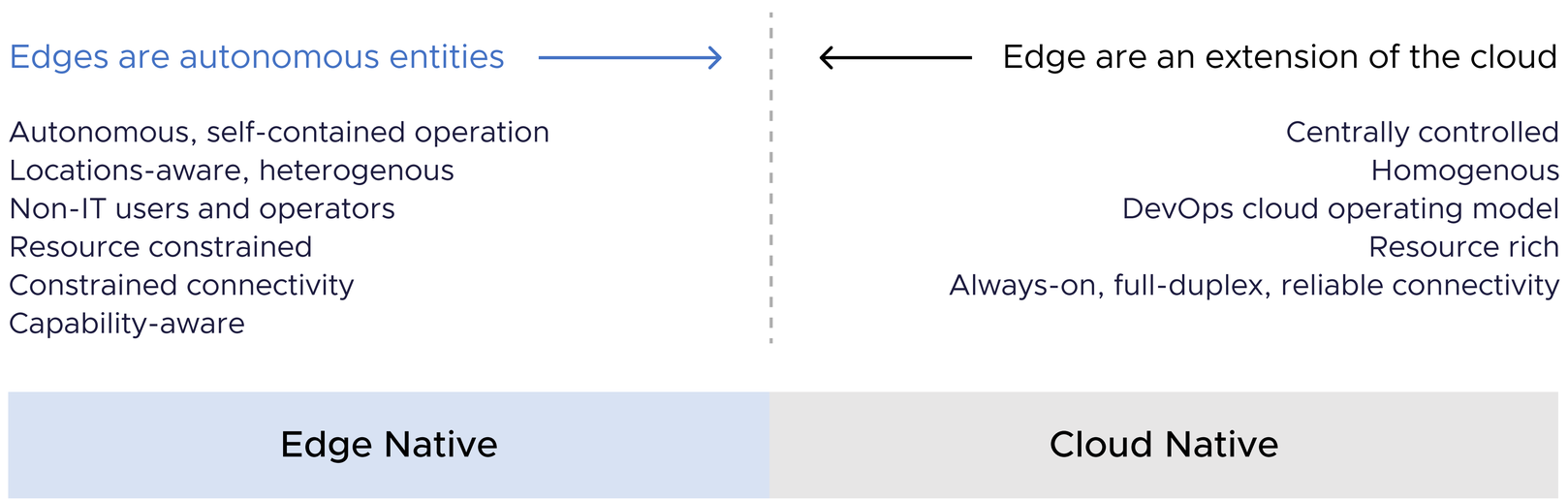
CONTENTS

EDGE NATIVE APPLICATION DESIGN

1

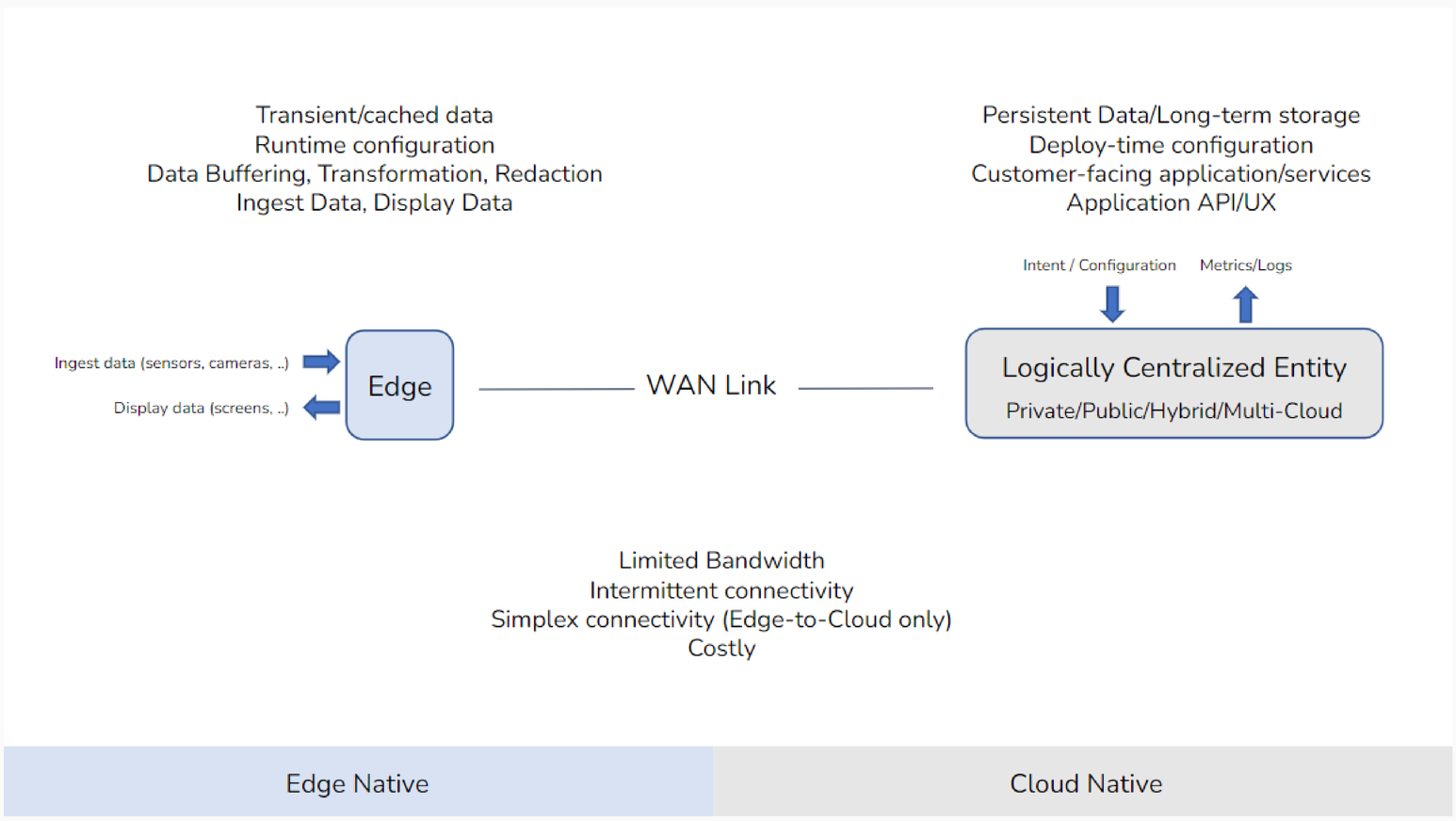
Edge Native and Cloud Native Application Design Domains

The diagram below gives an overview of deployments where “Edge Native” design behaviors apply.



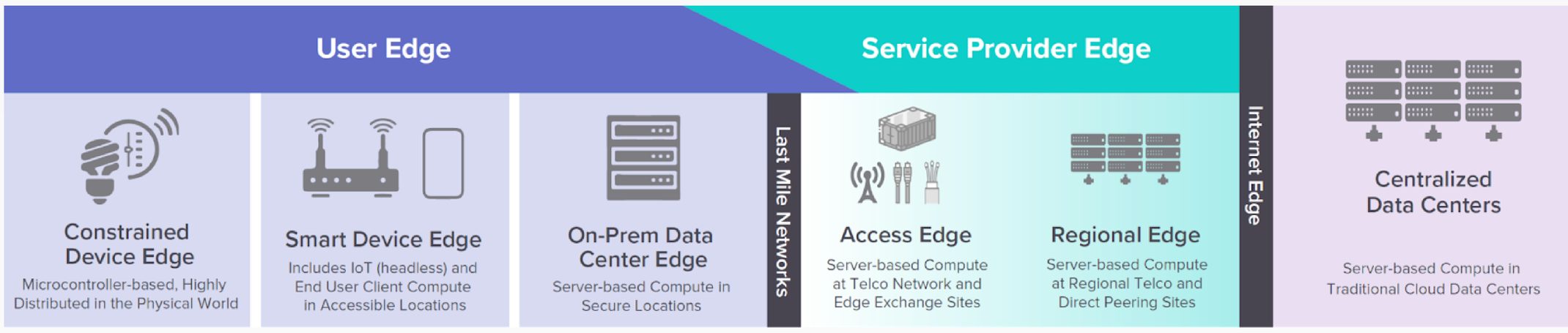
Edge Native Application Deployment Context

The picture below shows a reference application deployment that can be applied to many Edge native application scenarios. The application has a set of distributed Edge components and a logically centralized entity (often deployed in the cloud) where the distributed Edge components complement the logically centralized entity. The Edge components deal with functions that must reside at the Edge, such as bandwidth consumption reduction and adherence to location-based policies. The focus of the Edge components is often on ingesting data, local transformation of data, buffering of data, and displaying data. The Edge native design behaviors this paper outlines are targeted at these types of deployments.



The picture below shows a typical Edge native deployment context. Distributed Edges (with only one Edge depicted) are connected over a wide area network connection to a logically centralized entity. The Edges are interfacing with local devices to ingest data (e.g., from sensors) and to display data (e.g., screens). Overall configuration/intent is inserted into the system via the logically centralized entity. Metrics and logs are retrieved from the logically centralized entity. The picture also describes typical qualities of the Edges and the logically centralized entity which are further discussed below as part of the application design behaviors.

The logically centralized entity can be deployed in many different ways – across different tiers and different organizational entities. This can be seen as a continuum of the Edge, as depicted in the Linux Foundation (LF) Edge Whitepaper "Sharpening the Edge: Overview of the LF Edge Taxonomy and Framework".



Entities to consider, as referenced within the LF Edge continuum:

- Centralized Data Center (CDC / hub / parent)
- Service Provider Edge (ESP / intermediary / parent or child)
- User Edge (UE / spoke / child / peer)

Generally, outside of centralized data centers, Edge native principles and behaviors are applied. In other words, constrained environments are possible across the User Edge to Service Provider Edge continuum. Overall the properties of the resources and their associated policies create the relationship to classify something as 'Edge native' more so than the type of Edge (like Constrained Device Edge, Smart Device Edge, etc.) or the locality of an Edge deployment (like sovereign, single location stand-alone deployment, regional deployment, or multi-regional deployment).

EDGE NATIVE CONSTRAINTS

The bulleted list below defines the categories of design constraints that typically need to be considered when designing an Edge native application.

- Connectivity constraints (i.e, data in transit or network constraints) include limited bandwidth, intermittent connectivity ("air gapped mode"), and delay and jitter between the Edge and the Cloud or other Edges. In addition, connectivity constraints can include any policy/security related rules or transformations that apply to any data in transit. For example, certain use cases require that data be anonymized and personally identifiable information (PII) be removed before it is transferred to another location. Connectivity constraints can also include limitations imposed by middle-boxes performing network address translation or packet filtering.
- Data at rest constraints include security or policy requirements and the associated rules that require certain data transformations and/or storage to occur at a specific location, like in a region or country.
- Resource constraints include limitations due to the resources available at an Edge, such as power, memory, space, and compute capacity.

The different types of constraints can generally apply to the different design behaviors detailed in the following section, but the applicability can vary by deployment.

EDGE NATIVE APPLICATION DESIGN BEHAVIORS

2

"Edge native" application design means evolving and complementing the Cloud native application design methodology, such that applications can deal with the constraints of the Edge. This paper considers the **12-factor methodology** as a reference methodology for cloud native application design. The basic principles of Cloud native application design, such as the separation of data and code, the notion of processes as stateless, share-nothing entities, and the separation of build and run stages, are applicable to the Edge as well. The design behaviors listed below are those that are evolved, reconsidered, or even new when designing an Edge native application. This list should serve as a reference for developers building Edge native applications.

CONCURRENCY / SCALE

As defined by the CNCF IoT Edge WG "**Edge Native Application Principles Whitepaper**", Edge native applications are "spanning". They can scale geographically to span multiple Edges or scale by processes within an Edge to span multiple failure domain boundaries.

- **Scale *across* locations or Edges**: Apps are deployed to "Edges," with each "Edge" being one or more compute nodes. An "Edge" represents a physical or logical grouping of compute nodes that run one or more applications and are composed of small multi-purpose compute nodes. There can be many Edges. Instances of the same App can be deployed to many Edges simultaneously.
- **Scale *within* a location or Edge**: Apps are designed so that they can benefit from running on several nodes in parallel (12-factor methodology calls this "scale by the process model"). Resources needed within an Edge are met by leveraging a group of small, low-cost devices, rather than a single large compute unit. Grouping/clustering multiple devices also allows leveraging of high-availability mechanisms for the Edge-local control plane (e.g., Kubernetes control plane high-availability mechanisms could be used if a Kubernetes cluster is used at the Edge). Multi-architecture (x86, ARM) ensures that different use cases can be met. Entire Edges can fail (as in all the devices at a location) – and would be re-bootstrapped using state that might be retrieved from a central hub (cloud hosted) when recovering from a failure.

EDGE AUTONOMY VIA DEPENDENCY AND POLICY MANAGEMENT

Declare all dependencies explicitly to allow Apps to operate even if connectivity to the cloud is lost.

- Explicitly declare and isolate dependencies and policies – Edge-local and cloud.
- Edges pull information from the peers or parents and post results accordingly.
- Edges operate autonomously – the declared state or intent for an Edge and the applications running on an Edge are declared on a parent or peer and retrieved accordingly.
- Edges decide autonomously about the role a node fulfills, the assignment or scheduling of jobs to nodes, etc.
- Edges continue to operate when disconnected from the parent or peer (within a location and across locations where and when possible).
- Edges can reach and access an entity to retrieve declared state or intent from time-to-time. This entity is typically hosted by a parent but could also be from a peer. No assumptions are being made whether connections from outside of the Edge towards the Edge (e.g., Cloud to Edge) can be established. For example, no parent initiated contact with an Edge or a persistent tunnel, or an "always connected mode" is assumed. This ensures the Edge is pulling configuration from the parent or peer. Pushing to the parent or peer is an option, but more difficult to secure and manage.
- Any backing service for applications and services running at the Edge are treated as an attached, potentially remote, resource.

DISPOSABILITY

Design with failures in mind: Nodes, Edges, and Apps are allowed to fail at any point in time.

- Robustness by Edge: Edges should be fast to start up and should gracefully shut down.
- Edges retrieve their declared state (which likely includes startup and runtime configuration) from a parent or peer. This parent or peer hosts the declared state or intent and can be cloud-hosted, including tiered deployment models that include intermediaries or Edge services providers on behalf of the central entity.
- If an Edge should fail, assurance should be made that the Edge is not ‘bricked’ or otherwise irretrievable, unless it contains sensitive data and is intentionally ‘bricked’ in a self-protective mode / posture.

CAPABILITY-SENSITIVE

Edge native applications should provision a means for Edges to be aware of their environment or deployment context.

Capability sensitivity, which means an application’s awareness of its environment or deployment context, i.e., the available (hardware-)resources and associated or attached devices, is another principle of Edge native applications. The **Edge Native Application Principles white paper** discusses those as part of the categories “hardware awareness” and “interaction with external resources”. Capability sensitivity isn’t typical for cloud-applications and as such is also missing from the 12-factor principles.

As explained in the **Edge Native Application Principles Whitepaper**, Edge applications are often aware of their capabilities in the form of hardware, external devices, and network availability. More specifically, they should be dynamically ‘knowledgeable’ of their internally and, if possible, externally available network bandwidth, compute power, compute type (CPU, GPU, etc.), sensors, cameras, actuators, users, and more. Without this ‘knowledge’ it is not possible to create common configurations that can be applied at scale while allowing customization during install or runtime by using local environment values as configuration input.

DATA PERSISTENCE

It is recommended that any data that needs to persist should be stored in a dedicated, stateful backing service. Wherever and whenever possible, Edges should be stateless and buffer / cache data only. The more stateless an Edge application, the more it can fulfill the Edge native application principle of being portable and reusable.

- Depending on the properties of the data that require storage (retention, sensitivity, size, etc.), an Edge native application may persist data locally on the Edge location. For data that is actionable, it may be necessary to surface the data to the parent or peer and may be removed from the Edge location.
- Data / state (configuration, customer data, etc.) is typically persisted in the parent or peer in a dedicated, stateful backing service. Edges may cache or buffer data for local processing needs and in the case of disconnected operation. Large amounts of data are typically not persisted at the Edge long-term (i.e., preferably store data and configuration in the parent or peer).
- Data originating at an Edge is typically posted to the control plane at regular intervals to allow the control plane to infer when the Edge is offline and not expect data from the Edge in a sequence or time-series.
- An acceptable level of time synchronization should occur at the Edge to stamp the data accurately for fitment in a time-series upstream.

(continued on next page)

- Data should be stored during times of disconnection of the Edge and be forwarded upon reconnection.
- Provisions should be added to protect the central entity (i.e., control plane) that keeps the declared state or intent for all Edges and that could also serve as a consolidation point for data posted by the Edges. The protection includes provisions, so that a large set of formerly disconnected Edges all trying to contact the cloud at the same time does not create a DoS attack on the control plane.
- When an Edge is in disconnected mode, it should limit storing aging data to be considerate of local storage availability. Allowing data to age and be discarded, filtered, or summarized to preserve limited storage is preferable.

METRICS / LOGS

Like cloud native applications, Edge native applications should be as centrally observable as possible. Treat logs and metrics as on-demand streams, use metrics whenever feasible.

- Stream or push metrics and logs to a parent when possible; logs are only made available on-demand (to save bandwidth between Edges and the parent).
- Focus on “actionable” metrics: it is preferable to send “information”, not “data”.
- Log data at an Edge is typically posted to the control plane at regular intervals to allow the control plane to infer when the Edge is offline and not expect data from the Edge in a sequence or time-series.
- Log data should be stored during times of disconnection and forwarded upon reconnection. The considerations to protect a central log / metrics server mentioned above for “storage” apply here.
- When an Edge is in disconnected mode it should limit the storage of aging log data to be considerate of the constrained storage available. Allowing log data to age and be discarded, filtered, or summarized to preserve limited storage, is preferable. This is very much related to the “actionable metrics” instead of sending “information” points from above also.

OPERATIONS (OF EDGES AND NODES)

Assume a non-expert operator for Apps. Metrics / logs exposed by an App should be actionable.

- Assume a non-expert operator. An IT skill set should not be assumed for on-site personnel at an Edge location.
- Edges and nodes are disposable and should tolerate starting or stopping at a moment’s notice.
- Neither Edges, nodes, nor the processes that they run, are “debugged” in case they do not function properly: “Factory reset – re-bootstrap” to resolve issues at an Edge.
- Applications are typically compatible with Zero-Touch provisioning guidelines to avoid misconfiguration.
- Green-booting (aka rolling back) to last-known-good-state can also help resolve issues and allow operations to resume while logs are used to determine why an update or upgrade failed.
- **Configuration**
  - Configuration state for applications and Edges is typically persisted in a logically centralized entity (typically a parent; tiered or distributed deployments, apply for this logically centralized entity such as intermediaries or Edge service providers)
  - Startup configuration or parameterization for Apps should be generic for all Edges. Runtime configuration on a per Edge basis is retrieved by the App at runtime from the logically centralized entity.
  - Whenever and wherever possible, attempt to develop applications so that they require a limited amount of configuration data and can run well with default values wherever they are deployed. This allows for simple, cookie-cutter style deployments across many Edges.

EDGE NATIVE APPLICATION SAMPLE SCENARIO

3

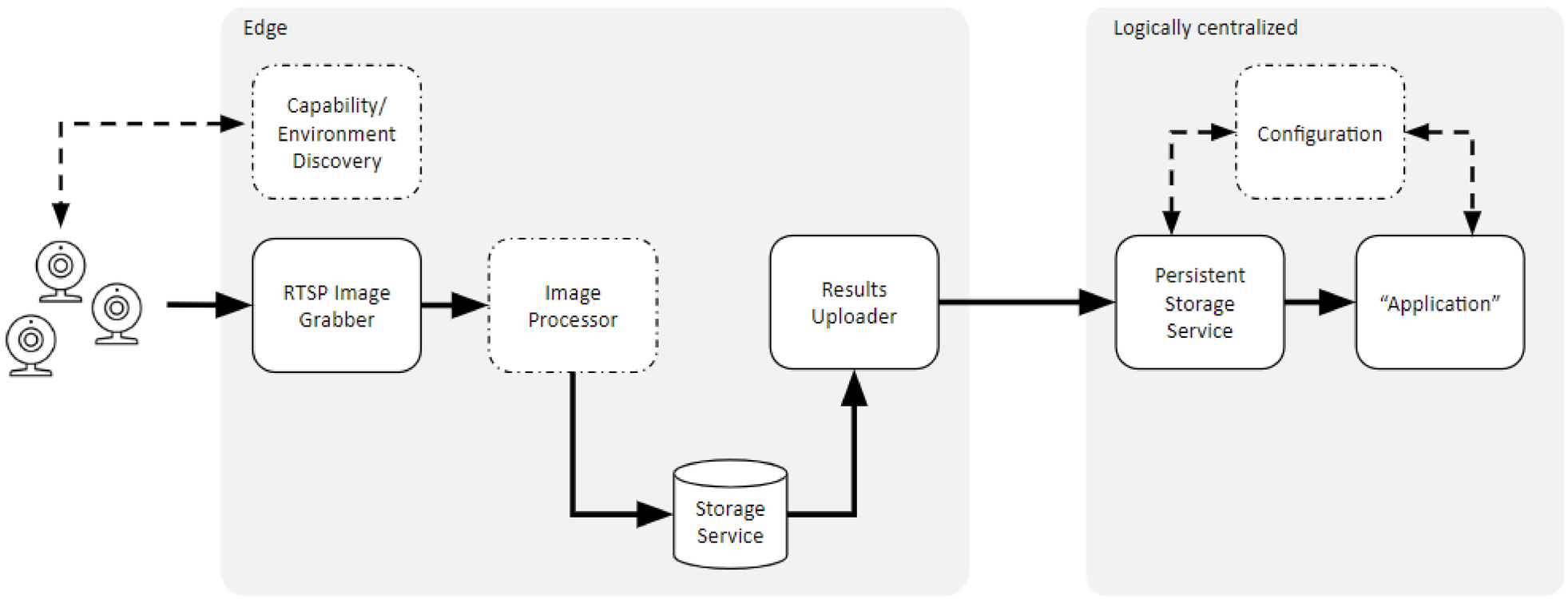
Reliable video capture and transformation (VCAT) is a common component of many Edge use cases, such as surveillance for security or compliance purposes, or customer behavior and footfall analytics. These use cases can be found in many verticals, such as retail, quick service restaurants, transportation, logistics, etc. One or more cameras capture video that needs to be stored for later reference or processed to extract information from the video. The connection between the location of the cameras and the cloud is limited so that the camera streams cannot be sent to the cloud for processing or storage. The connection may not have the necessary bandwidth, it may be too expensive, or it may be subject to outages, making upload to the cloud unreliable. Uploading to the cloud may also not be feasible because of security or policy reasons. For example, say images sent to the cloud cannot contain Personally Identifiable Information (PII). In addition to the video-related constraints, the Edge sites may face additional connectivity constraints due to the network architecture and the associated network security.

EXAMPLE SOLUTION

An example solution for reliable video capture and transformation (VCAT) would offer the following capabilities:

- Receive and sample video streams from multiple cameras.
- Process the received frames. Processing can include multiple transformations, such as:
  - Sampling the incoming frames at a specific, user-defined frame rate.
  - Resizing the images to a user-defined size.
  - Detecting and marking objects in the images.
  - Removing PII from the images (e.g., blurring faces).
- Buffer / cache the images locally in the Edge and upload the results of processing them either immediately, upon connection from the Edge to the cloud, or at user-configured times.
- Persist images in the cloud (in long-term storage).
- Further process the images uploaded to the Cloud by an application. This can be any type of application that implements a specific use-case, such as surveillance, analytics, etc.

The following figure outlines the architecture of a sample solution, including the functions that reside at the Edge and the functions that reside in the Cloud.



The figure shows a scenario of a VCAT application implementation where an application wants to process the video streams from three cameras. The edge part of the application can include a component that automatically discovers the cameras and their capabilities (see the Capability/Environment Discovery box surrounded by a dotted line). The edge application includes an RTSP Image Grabber component that receives the video stream and converts the stream into a sequence of images. An "Image Processor" component further processes the images (for example, it could redact the images and remove PII). After processing, the images are passed to a local caching service. The Results Uploader component retrieves the images from the local storage service and sends the images to the application's logically centralized entity. The local storage service decouples data ingestion from data upload. If the Edge is not connected to the logically centralized entity, images are buffered by the local storage service. The logically centralized entity receives images from the Edges and stores them in a persistent storage service. Additional uses of the images, as well as the configuration of the overall setup, are abstracted into generic components, shown as "Configuration" and "Application".

HOW DOES EDGE NATIVE APPLICATION DESIGN BEHAVIOR APPLY?

Edge native application design behavior applies to the VCAT-Edge and VCAT-Cloud portions of the VCAT application as follows:

- Concurrency / Scale: The Edge portion of the VCAT solution (VCAT-Edge) is deployed on a set of compute nodes located at each Edge site. Each Edge site is deployed with the same deployment configuration.
- Dependencies / Policies / Edge Autonomy: VCAT-Edge is designed to continue operating if the connection to the cloud is lost. If the connection is lost, the fact that images are always written to a local volume first ensures that video streams continue to be captured and information is not lost.
- Disposability: If a VCAT-Edge instance fails, it automatically restarts and retrieves the required deployment configuration from the VCAT-Cloud instance. Because data is persisted in the VCAT-Cloud instance, only the data cached at the Edge is lost in the event of a failure.
- Capability Sensitive: Each VCAT-Edge instance would “discover” its local environment, i.e., determine the IP-address of the RTSP-endpoint that the camera represents. Local capability or environment discovery means that the VCAT-Edge instance automatically adapts its behavior to the local environment, rather than requiring explicit configuration. Explicit configuration would be difficult to manage in large, constantly changing environments.
- Data Storage: As already discussed above, VCAT-Edge instances only cache / buffer data.
- Metrics / Logs: In a production environment, the VCAT-Edge offers a set of metrics that are actionable by a non-expert operator. These metrics / logs indicate for example, whether the VCAT-Edge application or the entire Edge needs to be restarted (i.e., an action that could be taken by a non-expert operator).

Operations: VCAT-Edge is designed to require no site-specific configuration – at least no site-specific configuration that could not be automatically configured, e.g., through discovery operations, API calls, etc.

CONCLUSION & NEXT STEPS

This paper is a first edition and may experience revisions. Future papers related to sub-sections of this paper are anticipated. To reach out with feedback, collaboration requests, or questions, please create an issue on the **CNCF Runtime TAG GitHub page**.

HOW TO GET INVOLVED

The CNCF IoT Edge Working Group has regular meetings, a mailing list, and a Slack. See the **Communication section** of the working group GitHub page for the most up to date information. We welcome the reader to get involved by presenting Edge related projects, bringing an idea for an area of work for the group, or helping to revise this whitepaper and/or draft a follow-up paper.