Releases, Versions & Patches

This policy was approved by the TSC on 1/16/19

This policy was updated with TSC approval on 6/25/19.

This policy was updated with TSC approval on 7/26/19 to add DS/AS and associated SDK version/release strategy.

Release Cycle / Cadence

The EdgeX Foundry community publishes new releases of EdgeX on a regular cadence, enabling the community, businesses and developers to plan their roadmaps with certainty of access to new features.

Releases of the collection of central or core EdgeX micro services get a development codename (example: 'Edinburgh'). EdgeX micro services that are part of this collection include:

- Core Data
- Metadata
- Command
- Registry/Configuration (Consul)
- Logging
- Scheduler
- Notifications
- Rules Engine
- Export Client and Distro Services
- Security Services
    - API Gateway (Kong)
    - Secret Store (Vault)
- System Management Services
- Additional supporting infrastructure for these services
    - Database (Mongo or Redis today)
- Bootstrapping/initialization services
    - Configuration (config-seed)

For the purposes of the rest of this document, the term "EdgeX release" refers to the release of this collection of central EdgeX micro services. The EdgeX release code name is sequential in that the code names are alphabetical. The Edinburgh release is before the Fuji release, which is before Geneva release, etc. The EdgeX release code names are based on geographical locations. These releases also get a version number. For example, the Delhi release was version 0.7. EdgeX releases the collection of central or core EdgeX micro services twice a year (typically around April and October). Additionally, all microservices in the central EdgeX collection (listed above) have the same version number corresponding to that release. For example, both the Core Data microservice and Core Metadata microservice for the Delhi 0.7 release will have version numbers of 0.7. For a patch release (explained below) such as 0.7.1, all of the microservices have the same version number 0.7.1.

EdgeX's versioning scheme is based on semantic versioning (https://semver.org/). However, the project may, at times, decide to release a major version due to significance-based changes or marketing decision.

The EdgeX codenamed release should not be confused with MAJOR version! Per semantic versioning definition, a MAJOR version is typically a significant milestone in the product that includes key features or updates and may include non-backward compatible changes (features, APIs, etc). Not all of the EdgeX codenamed releases completed each six months will be "significant" or contain "non-backward compatible changes". See major, minor and patch version information below.

Independent Tools and Services Release Cycle/Cadence

The following section was approved by the EdgeX TSC on 7/26/19 by email.

Some EdgeX services (and tooling) have a separate release cycle. The SDKs and services stemming from the SDKs, for example, are not necessarily released with the EdgeX release. Which services are allowed to have their own release cycles is at the discretion of the EdgeX Technical Steering Committee (TSC). Today, the following tools and services are released on independent release cycles:

- Device Service SDKs (Go and C)
- Device Services
- Application Functions SDK (Go)
- Application Services

When appropriate, a working group chairperson requests permission to release these tools and services. The release manager, at the request of the working group chairperson, is responsible for the release of these tools and services. Again, these tools and services are versioned by semantic versioning just as the rest of EdgeX to indicate what is included with the release (and specifically whether it includes compatible or non-compatible features and APIs). Disagreements with a release manager decision about when to release is adjudicated by the TSC.

SDKs and Associated Services Major Releases

SDKs (Device or Application Functions) release major versions that coincide with the major release of EdgeX. To reiterate a major release is one that contains non-backward compatible changes. The major versions of SDKs, for example, are tightly coupled to the major version of EdgeX. At this time, the major versions of EdgeX and the SDKs will be the same (i.e. 1.x of the SDK works with 1.x of EdgeX, and 2.x of the SDK works with 2.x of EdgeX). It is the goal of the project that major releases of the SDKs generally release at or near the same time (within months) as the EdgeX major releases.

Services created from the SDKs - for example, Device and Application Services - are versioned from the SDK. Thus, major versions of the device and application services must be built/rebuilt when a new major release of the SDK is created.

SDKs and Associated Services Minor Releases/Patches

SDKs (Device or Application Functions) can release minor versions or patches completely independently of the EdgeX release (and vice versa – minor releases of the rest of EdgeX can occur independently of SDK minor releases). A minor version or patch is backward compatible (with regard to APIs, configuration, etc.) with the existing major version of the SDK. For example, after 2.0 major release of EdgeX and the Device Service SDK, the SDK may choose to release versions 2.0.1, 2.0.2, 2.1.0, 2.2.0 and 2.2.1 while the rest of EdgeX is still at version 2.0. All of these minor releases would still be compatible with EdgeX 2.0.

Conversely, EdgeX may have a 2.1 minor release, and this would not impact the Device Service SDK. Any 2.x version of the SDK would still work with any version of EdgeX 2.x.

Device and Application Services are again versioned from the SDK. Thus, device and application services with a minor release version (for example 2.0.1 or 3.1.0) indicates the device or application service was built from the associated SDK version (in this case 2.0.1 and 3.1.0 respectively).

Version Query

All services will have a “ping” API that will respond with the version number. This allows independent services and tooling (listed in the previous section) to query other EdgeX services that it works with and check for compatibility. For example, when a Device Service built to work with EdgeX 1.0 comes up and it queries Core Data via the ping API and gets a response of 2.0, the device service can choose to log the issue (providing a clear indication of a version mistach) and exit. The query calling service determines whether it is compatible or not (exiting or continuing). Each services only offers a query API that indicates what version of EdgeX it is.

Major, Minor Versions, Patches and Pre-release Versions

Again, EdgeX follows semantic versioning. Major releases have a “major number” or version number like 1.0, 2.0, etc. Major releases will typically include significant new features, new or significantly updated services, new or significantly updated APIs, etc. and may be incompatible with previous releases (major or minor). Minor releases may contain some new or updated functionality but should always be backward compatible with the associated Major and Minor releases it is based on. Minor releases have a version number which includes a Major release number, a decimal, and a minor release number (like 1.1, 2.1, 2.3, etc.). Patches (or Patch releases) are releases that contain backward compatible bug fixes. Patches should not contain any new or updated features except those necessary to address the bug(s). Patch versions are based on a major version number, a minor version number (or zero if the patch is to the original major version, and a patch version (like 1.0.1, 1.2.1, 2.1.1, etc.) with decimals between these numbers. EdgeX has automated pre-release versions with suffixes to speed up integration and testing between the different components. These suffixes are, “dev.X” (where X is a number) to denote a developmental release and “rc.X” to denote a release candidate.

Pre-release Versions and the Release Cycle

Automated pre-release versions are released during the release cycle. During normal development time the releases will be marked with the suffix “-dev.X” (where X is a number) denoting a developmental release. Once EdgeX has entered a code freeze the automated releases will be tagged with the suffix “-rc.X” (where X is a number) to denote a release candidate. For components that have their own release cycles, the working group chair should decide when code freeze starts. When a component is ready for release, developers should inform the working group chairperson that manages the component. It will be up to the working group chairperson and the release manager to decide when to cut the release for the component. Once the release is cut the version will be updated to the next semantic version and the suffix will revert to “-dev.X” until the next code freeze.
The “next” semantic version can increase the major, minor or patch version of the component. If the TSC has already decided on the version number for the next named release (Ex: Delhi is 0.7 and Edinburgh is 1.0) the component should be set at that version. If the TSC has not decided on the version number for the next named release, then the working group chairperson and release manager can increase the patch version. It is recommended to wait until the TSC has decided the next version to release the component. However, for certain components that are released on their own release cycle (ie: SDKs) this may not be viable. For core releases during code freeze EdgeX will cut a release branch. For each repository, once the named release branch is cut then the master branch will be increased to the next forecasted version. From this point forward, the release branch will only be allowed to increase its patch version to not break the semantic versioning.

Example

A repository has a branch cut for the Edinburgh release called “edinburgh”. The edinburgh branch will be version 1.0.0-rc.X and master will be 1.1.0-dev.X. This is because that master is now tracking development for the Fuji release. Similarly, the edinburgh branch is tracking release candidates for the Edinburgh release. Once the Edinburgh release is finalized the edinburgh branch will versioned at 1.0.0. The next version for edinburgh branch will be set to 1.0.1-dev.X to allow for any small patches like bug fixes or security patches.

Versioning and the Release Cycle

At the conclusion of each EdgeX release, the TSC meets and forecasts the version (major or minor) for the next release cycle. Based on the functionality to be implemented and included with the upcoming release, the EdgeX TSC may choose to preliminarily version the next release as a major release. If the next release will contain only backward compatible changes and deemed not to contain a significant amount of new and updated features, the TSC may choose to version the upcoming release as a minor version. For example, if the current release – codenamed "Portville" was a major version 7.0, the TSC may choose to set the next release to major version 8.0 (if they plan to include non-compatible changes or if the amount of new and updated functionality is significant enough to want to distinguish the release as a major release) or as a minor version 7.1 (indicating no changes that break compatibility with the prior named release).

All the VERSION files (on the current Master branch of the repositories) along with any other relevant indicators in the project repositories will immediately be updated to indicate this TSC decision.

Some event during the release cycle could dictate the TSC to change the version decision made at the start of a release. This could be from major to minor or from minor to major release. A decision to change the version strategy in the middle of the release cycle is not one that is done casually, should be avoided at all costs, and will cause a certain amount of hardship on the developers and confusion to users (example: development artifacts in repositories like Nexus or Docker would suddenly change and make it look like a new release was created). Below are some examples of when the TSC may change the version decision:

- When an unforeseen technical challenge arises and the best technical solution as decided by the TSC requires non-backward compatible change (forcing the next version to be a major version vs minor).
- When, nearing the completion of the release cycle it is discovered that compatibility remains yet it was expected that the release would require a non-compatible change (allowing under some circumstances for the TSC to return to a minor version vs major)
- When an expected feature associated with the release is not going to make the release deadline and causes the TSC to re-evaluate the versioning (again more likely going from major to minor)
- The TSC should not generally make changes to the release/version strategy during the release cycle due to non-technical issues or concerns (example a marketing request to make a major version). The incorporation of non-technical issues affecting the version decisions should only be taken into consideration during release planning.

Release Manager & Patch Decisions

On the TSC's decision to forecast the upcoming version for the next release cycle, the management, coordination and orchestration of the release cycle is turned over to the release manager.

Decisions about patches (versions made between the release cycle that are used for for backwards-compatible bug fixes) are at the discretion of the release manager for that release cycle. A work group chairperson can make the request for a patch release, but the decision lies with the release manager. Disagreement can be appealed to the full TSC for adjudication. The release manager must coordinate the patch release with all applicable working groups (via the chairman) - thus the reason a working group chairperson is not unilaterally allowed to produce a release.

The release manager can coordinate "pre-release" versions of the artifacts in the middle of the release cycle. These artifacts are used by the developer and user community to demonstrate and test upcoming release functionality and changes. These artifacts and source code use a version string containing a hyphen followed by an arbitrary string (like -dev, -test, -{timestamp}, etc.) to distinguish them from the final release of that version (see paragraph 9 of the semantic versioning specification for more background).