Milestone App Center - Developer Guide

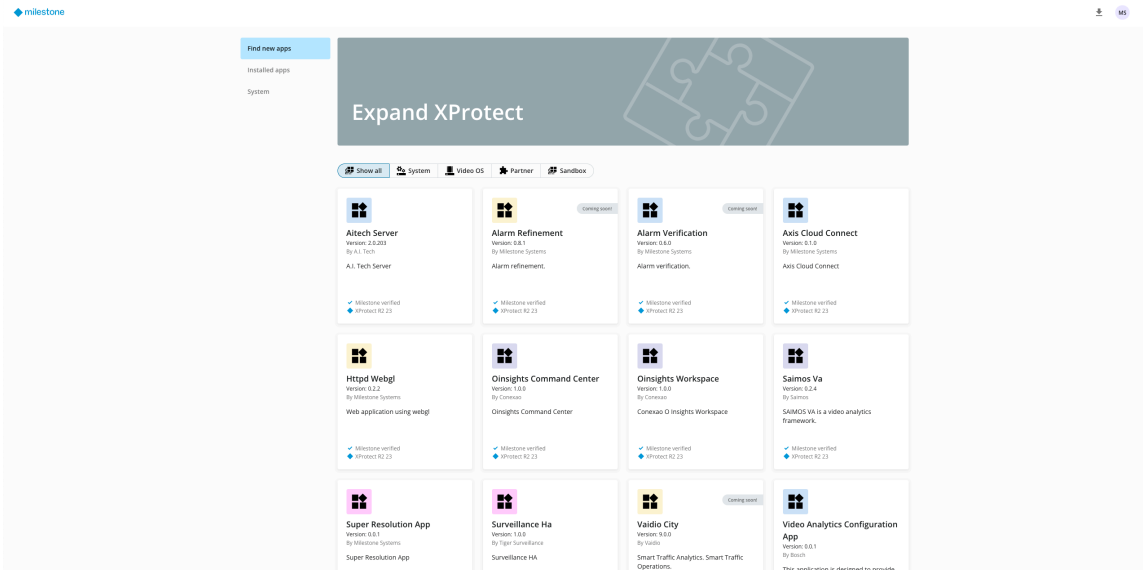
> *Build and deploy applications for the Milestone App Center.*
>
> *This guide helps you:*
>
> - *Understand the platform and tech stack*
> - *Set up your development and cluster environment*
> - *Package and register an App for installation*
> - *Reuse and learn from the official sample apps (including sandbox usage)*

1 App Center

Milestone's **Open Platform** has always been the backbone of XProtect's flexibility—enabling seamless integration with a wide array of hardware, software, and third-party services. Now, that openness is enhanced with a more **streamlined and modular configuration experience**, giving technical users greater control with less complexity.

With just a few clicks, you can deploy your entire system, including access to the **App Center**, where you select the exact services that define your ideal video management setup. From trusted video analytics to specialized integrations, you can unlock the full value of your video data and convert it into actionable intelligence. All components—core VMS features, analytics engines, and third-party apps—are managed from a **single interface**, where installation and updates are handled with one-

click operations. This centralized approach reduces deployment time, simplifies maintenance, and ensures consistent performance across your environment—whether you're managing a single site or scaling across a distributed enterprise.



2 Tech Stack

The runtime environment powering the App Center is built on a robust and scalable cloud-native tech stack designed for flexibility, observability, and performance.

2.1 Kubernetes (K8s)

An open-source platform for automating the deployment, scaling, and management of containerized applications. Kubernetes provides a unified control plane that enables efficient orchestration, high availability, and horizontal scalability—making it essential for modern microservices architectures. 🔗 Learn more

2.2 Helm

A package manager for Kubernetes that simplifies the deployment and lifecycle management of applications—used here for packaging and deploying Apps. Helm Charts allow you to define, install, and upgrade even the most complex workloads with minimal effort. 🔗 Learn more

2.3 Istio

A powerful service mesh used for:

- Implementing the Kubernetes Gateway API to manage ingress traffic with TLS termination and routing
- Enforcing authentication and authorization policies across workloads (ingress + in-cluster access control)
- Enabling secure service-to-service communication via mutual TLS (service mesh security)

2.4 Prometheus

A monitoring system that collects metrics from configured targets at defined intervals—used here as the metrics backbone feeding Grafana dashboards. It supports rule-based evaluation and alerting, making it ideal for tracking system health and performance.

2.5 Grafana

An open-source analytics and visualization platform that integrates with Prometheus to provide real-time insights into infrastructure and application metrics through customizable dashboards—forming the dashboards part of the monitoring pair.

2.6 Apache Kafka

A distributed event streaming platform optimized for high-throughput, real-time data ingestion and processing. Kafka supports:

- Publish/subscribe messaging
- Durable, ordered storage of event streams
- Real-time stream processing

Kafka is used here for real-time data streaming, enabling resilient pipelines and reactive applications that operate on both historical and live data.

This stack ensures the App Center is scalable, observable, and ready for dynamic workloads in production environments.

3 Setting Up the System

To develop and deploy an application in the **Milestone App Center**, follow this setup process. Once the cluster is running, your app can use the built-in tech stack for deployment, monitoring, streaming, and secure communication.

3.1 Set Up the Runtime Environment (Installation Wizard)

To deploy applications in the **Milestone App Center**, you first need a working environment for running containerized workloads. Here's what you need:

- An `XProtect installation` .
- A **client machine** to run the `installation-wizard` wizard. (Open the How-to-Guide.pdf in that folder for full step-by-step instructions.) This installs and configures the cluster on the target Linux machine. Common optional flags:
  - `--cluster-sandbox` : Set up a local sandbox consisting of a Docker registry and Helm repository allowing you to deploy and install your app locally, before submitting the app.
  - `--cluster-dev-mode` : Enable Kubernetes & Helm dashboards for monitoring and debugging
  - `--cluster-debug` : Output verbose installation logs
  - `--dbg` : Run the UI installer in debug mode
  - `--update` : To speed up the installation, you can set this flag and it will skip whatever is already installed
- A **Linux target machine** (Ubuntu 24.04) with:

- SSH server installed:

  ```
  sudo apt install -y openssh-server
  ```

- A static IP address

Once setup is complete, the Linux machine will be running a Kubernetes cluster with all required runtime components. You can access the App Center via:

- Directly at: `http://<system-ip>/app-center`
- Or using the **Management Client** under the App Center node by installing the `AppcenterInstaller.msi`

📘 For detailed instructions, refer to the How-to-Guide of the installation wizard.

3.2 Setting up a Dev Machine

Once the installation is complete, set up your development machine to connect and work alongside the cluster.

> ***Windows Users:*** *If you're developing on a Windows machine, you'll need to install and configure Windows Subsystem for Linux (WSL) first. See Appendix A.2 for detailed setup instructions.*

3.2.1 Running on Linux

To communicate with and develop an app for App Center, make sure you have the following tools installed:

- **Docker**
- **Helm**
- **kubectl**

You can install them using the `setup.sh` script which installs and configures **Docker**, **kubectl**, and **Helm**. To execute the script run `chmod +x setup.sh` to make it executable and run:

```
./setup.sh
```

The script will prompt you for the **IP address** and **SSH credentials** of a server in your system. If you’re running a multi-node setup, select the server designated as the **master node** in your Kubernetes cluster.

Once the setup completes, close the terminal and open a new one to ensure your environment variables are properly reloaded. To confirm the connection was successful, run:

```
kubectl get nodes
```

This command lists all nodes in the cluster. Example output:

```
> kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
cluster-name   Ready   master   1h   v1.29.2
```

The cluster is now up and running, providing all the essential components developers need to start building and deploying applications.

3.2.2 Accessing the Kubernetes and Helm Dashboards

To troubleshoot and monitor the internal state of your system, both the **Kubernetes Dashboard** and **Helm Dashboard** offer valuable insights. These tools provide visual interfaces to inspect cluster resources, track deployments, and identify potential issues quickly.

To access the Kubernetes Dashboard, run the following Helm command

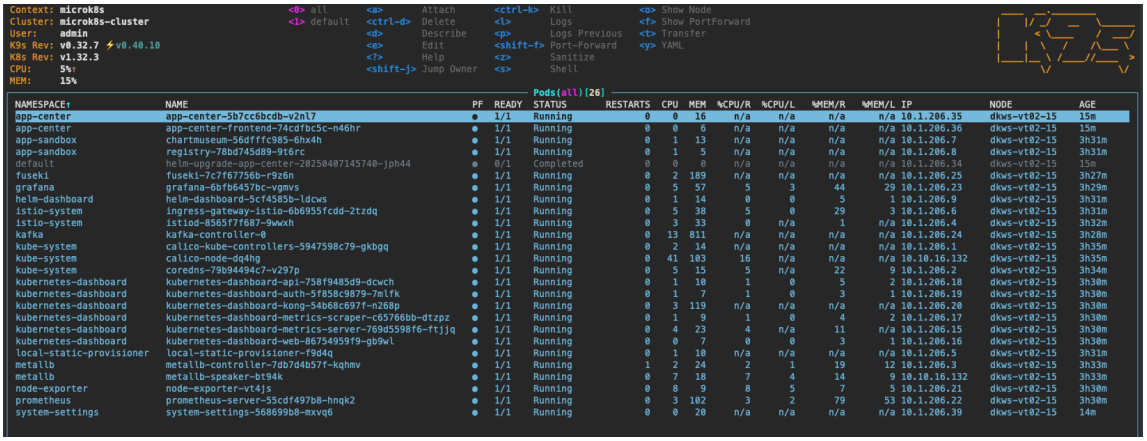
```
kubectl create token default | xclip -sel clip; kubectl -n kubernetes-dashboard
port-forward svc/kubernetes-dashboard-kong-proxy 10443:443
```

This will first generate an access token, copy it to your clipboard, and then port-forward the Kubernetes Dashboard service to `https://localhost:10443/`. The dashboard remains accessible as long as the port-forwarding command is running. When you open the page, you'll be prompted to enter the token—simply paste it from your clipboard. Note: the token expires after one hour, so you'll need to re-run the command to regain access.

The Helm Dashboard is available at `https://<system-ip>/helm-dashboard/`.

> ⚠️ *Note: Both the Kubernetes and Helm dashboards are only available if the system was set up using the `-dev-mode` option.*

Another useful CLI tool for monitoring your cluster is K9s, a terminal-based UI that helps you interact with Kubernetes resources efficiently:



3.2.3 Developer Options – Sandbox

The *sandbox* lets you test dev builds without publishing them externally. It runs a local **container registry** and **Helm repo** inside the system:

- **Container images:**
  - Registry exposed at `<system-ip>:5000`
  - Images must be pushed under `sandbox.io/` (required for them to load).
  - Example: if system IP is `10.10.16.15`, push to

```
10.10.16.15:5000/sandbox.io/<image>:<tag>
```

- **Helm charts:**
  - Repo exposed at `http://<system-ip>/app-sandbox`
  - Example: list charts with

```
helm repo add sandbox http://10.10.16.15/app-sandbox
helm search repo sandbox
```

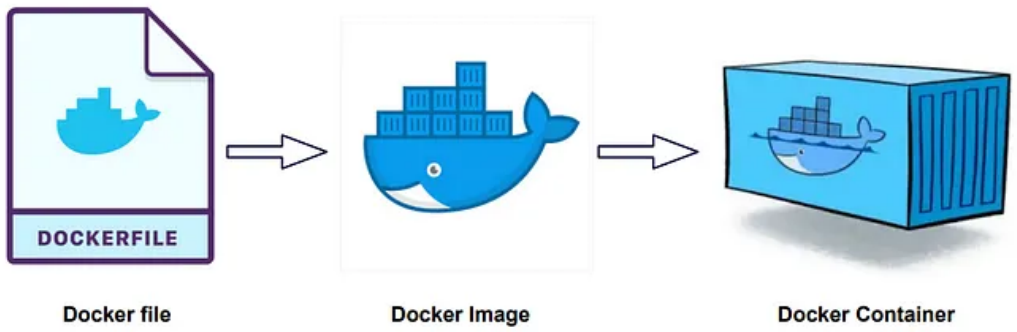
The sandbox is **disabled by default**. Enable it with the `--cluster-sandbox` option when setting up the system.

📘 For detailed sandbox workflow and app-builder integration, see App Builder documentation - Section 1.3.

3.3 Package your application

To run an application on the K8s cluster, you first need to package it into one or more containers. A container is a standalone executable that bundles everything required to run your app: code, runtime, system tools, libraries, and settings. For example, one container might serve the frontend, another an inference model.

Docker is the standard tool used to build these images.



Once the app is containerized, you share it by pushing the image to a **container registry** (e.g., Docker Hub, ECR, GCR, or a private registry). The K8s cluster is configured to pull from specific registries.

Examples of registry namespaces:

1. **System Repository** – Core system images
2. **VideoOs Repository** – Business apps/images, e.g.: {dockerregistry}:{port}/videoos.io/team-name/app:tag
3. **Partner Repository** – For partners, tag as: {dockerregistry}:{port}/partner.io/partner-name/app:tag*
4. **Sandbox Repository** – (if enabled) for dev and test images

Cluster registries for development:

- **Docker Registry (in-cluster, DEV mode)**
- **Milestone Docker Registry** – private registry hosted by Milestone.

3.4 App Builder

The **app-builder** tool helps you develop and package a containerized application that can be installed via the App Center. It takes an App definition file as input and generates a deployable package compatible with the App Center.

Note: Building the actual container images used by the App is outside the scope of this document. Basic familiarity with container image workflows is assumed.

For more details, visit the App Builder documentation.

3.5 Deploying an app to Milestone App Center

Use **App Builder** to package and publish your App. It automatically:

- Adds the `app-registration` dependency
- Sets category (partner / sandbox)
- Enforces namespace isolation (release name == namespace)
- Injects system template values (`system.ip` , `system.name` , `system.uuid`)

You normally do not edit Helm manually. For a minimal view of what the generated chart must contain, see Appendix A.3.

3.6 Legacy templates

- `legacy.managementServer` : Hostname of the Management Server in XProtect
- `legacy.useTLS` : Whether the legacy system is configured to use TLS

You can access these templates using the `template` or `include` functions in Helm.

As an App, the following rules apply:

1. **Namespace isolation:**
    - The chart must be installed into a namespace that matches the Helm release name.
    - This ensures each App has a private namespace and avoids conflicts with other Apps.
    - Example Helm install command:

      ```
      helm install my-release my-chart -n my-release --create-namespace
      ```

      Here, `my-chart` is installed using the release name `my-release` , and everything is placed in the `my-release` namespace.
2. **Namespace labels:** After installation, the namespace will automatically have these labels:
    - `app-registration/name`
    - `app-registration/version`
    - `app-registration/category`

3.6.1 What APIs can I use to interact with XProtect?

The easiest way to integrate with XProtect in Windows while using the App Center is via **AIBridge**.

Milestone AI Bridge is a set of cloud-native APIs that connect:

- XProtect Video Management Software (VMS)
- Intelligent Video Analytics (IVA) apps running as containerized workloads

It forwards camera video streams from XProtect to your analytics app and lets the app send results back as analytics data (events, metadata, video clips). In short: stream out, insights back.

Setup requirements for AIBridge:

> ⚠️ *Important: Before you can install the Processing Server app, you must install the **App Center plugin** (file name: `AppcenterInstaller.msi`) into the XProtect Management Client. Download it from: `AppcenterInstaller.msi` (locate `AppcenterInstaller.msi`). After installation, restart (or reopen) the Management Client—an "App Center" node will appear, which you use to install the Processing Server app.*

1. (If required for AI Bridge) Install the **Processing Server plugin** in the XProtect Management Client: Download plugin.
   - Login in with your **my Milestone** account
   - In the **Product** field, enter: "**Milestone AI Bridge**"
   - In the **Version** field, enter: "**Milestone AI Bridge 2.0.2**"
   - The list will show "Milestone AI Bridge resources", click to download the zip file.
   - Extract the "aibridge_xprotect_plugin"
2. In the Management Client, open the **App Center** node (enabled by the App Center plugin) and install the **Processing Server** app

Alternatively, you can use the **XProtect VMS API Gateway**. A sample application is available in the appcenter-samples here: API Gateway sample

3.7 Helm Recipe

Helm is a package manager for Kubernetes that simplifies the deployment of complex applications by bundling all necessary resources into **Charts**—YAML-based recipes that describe how to run your app on a cluster.

With Helm, you can use existing charts for common services or create your own. To generate a new chart, run:

```
helm create <chart-name>
```

This will create a directory named `chart-name` with the following structure:

```
> helm create my-app-chart
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
```

```
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

- **charts**: Contains any dependent charts your chart relies on.
- **Chart.yaml**: Holds metadata about the chart—name, version, description, etc.
- **templates**: Includes Kubernetes manifest templates used to deploy your app.
  - **deployment.yaml**: Defines how your app is deployed, including replica count and container specs.
  - **_helpers.tpl**: Stores reusable template functions.
  - **hpa.yaml**: Sets up Horizontal Pod Autoscaler based on resource usage.
  - **ingress.yaml**: Manages external access via ingress rules.
  - **NOTES.txt**: Displays helpful install notes after chart deployment.
  - **serviceaccount.yaml**: Creates a service account with specific permissions.
  - **service.yaml**: Defines how your app is exposed inside the cluster.
  - **tests/test-connection.yaml**: Verifies successful deployment and connectivity.
- **values.yaml**: Contains default config values that users can override during installation.

Appendix

A.1 Sample Apps

You can find all the sample apps to help you get started with the cluster on app-center application samples

- Hello World
- .NET Webserver
- PostgreSQL
- Kafka Topics
- AIBridge & Prometheus
- API Gateway Webserver

This repository is structured in two main directories.

```
appcenter-samples
├── build
│   └── common.mak
├── README.md
└── src
    ├── aibridge-prometheus-sample
    ├── apigateway-webserver-sample
    ├── dotnet-webserver-sample
    ├── hello-world
    ├── postgresql-sample
    └── utils
```

```
        └── auth.ps1
    ...
```

In the `build` directory, you'll find the `common.mak` file, which defines shared commands for building and managing the samples. Each sample directory includes its own `Makefile` that imports `common.mak` to leverage these commands. Under the hood, `common.mak` uses the App Builder (find more info here), which provides all the necessary commands to build, push, and install your application.

To work with a sample, navigate to its directory (where the `Makefile` is located) and run `make build` to build the Docker image and Helm chart for that sample.

`Note:` For the common.mak targets to work correctly, each sample must follow a consistent folder structure. Specifically:

- Application source code should be placed in a containers/ subdirectory, and each container must include a Dockerfile.

Available Make Commands

Command	Purpose
Development	
`build`	Build Helm chart and Docker image
`build-image`	Create Docker image only
`bundle`	Package app and dependencies
Registry	
`push`	Push chart and image to repositories
`push-image`	Upload Docker image to registry
`remove-image`	Delete local Docker image
`clear-image-cache`	Clear cached Docker images
Deployment	
`install-from-file`	Install chart from local `.tgz` file
`install-from-repo`	Install chart from Helm repository
`uninstall`	Remove deployed chart from cluster
`remove`	Delete chart from cluster
`list`	Show available Helm charts
Cluster	
`login`	Authenticate with cluster
`dashboard`	Open deployment dashboard

`events`	Show recent cluster events

To better understand what each command does, you can dry-run the desired command. Each sample directory also contains an extended **README** file to help you better understand the specific application.

A.2 Configure WSL

Step 1: Enable Windows Subsystem for Linux (WSL)

1. Open **PowerShell** as Administrator:
   - Press `Win + X` and select **PowerShell (Admin)** or **Terminal (Admin)**.
2. Run the following command to enable WSL and install the required components:

   ```
   wsl --install
   ```

   This will:
   - Enable WSL2
   - Install the **default** Linux distribution (Ubuntu 24.04.1)
   - Enable **Virtual Machine Platform** and **Windows Hypervisor Platform** (required for WSL2)
3. Restart your computer when prompted.
4. **Start Ubuntu**
   - After installation, launch Ubuntu from the Start menu.
5. **Set up your user**:
   - Enter a username and password when prompted.
6. **Check Ubuntu version (install 24.04 if needed)**:

   ```
   lsb_release -a
   ```

   Expected (if already correct):

   ```
   Distributor ID: Ubuntu
   Description:    Ubuntu 24.04 LTS
   Codename        Noble
   ```

   If you see a different version:

   1. Install Ubuntu 24.04:

      ```
      wsl --install -d Ubuntu-24.04
      ```

      If that fails, open the Microsoft Store, search for "Ubuntu 24.04" and install it manually.
   2. Re-run the version check:

```
lsb_release -a
```

Confirm the output matches the expected block above.

A.3 Helm Chart Requirements (Optional)

> *Optional: Only for debugging or inspecting what App Builder produced. Do not hand-maintain these unless directed.*

Minimum required elements (App Builder adds these automatically):

1. `app-registration` dependency in `Chart.yaml`
2. `app-registration.category` in `values.yaml` (partner or sandbox)
3. Release name == namespace (enforced during install)
4. Template values available: `system.ip`, `system.name`, `system.uuid`

Example `Chart.yaml` (excerpt):

```yaml
apiVersion: v2
name: my-app
type: application
version: 0.1.0
appVersion: "0.1.0"
dependencies:
  - name: app-registration
    version: "1.3.0"
    repository: "https://horizonsystem.azurewebsites.net/system"
```

`values.yaml` snippet:

```yaml
app-registration:
  category: partner
```

Inspect (read-only):

```
helm show all my-app-0.1.0.tgz | less
helm template debug my-app-0.1.0.tgz > rendered.yaml
```

Return to App Builder for normal iteration to stay compliant.