Preparing hawkBit for Production Use

Contents

Introduction

The Apertis project has been experimenting with the use of Eclipse hawkBit[1] as a mechanism for the deployment of system updates[2] and applications[3] to target devices in the field. The current emphasis is being placed on system updates, though hawkBit can also be used to address different software distribution use cases such as to distribute system software, updates and even apps from an app store.

Apertis has recently deployed a hawkBit instance[4] into which the image build pipelines[5] are uploading builds. The apertis-hawkBit-agent[6] has been added to OSTree based images and a guide produced detailing how this can be used to deploy updates to an Apertis target[7].

The current instance is proving valuable for gaining insight into how hawkBit can be used as part of the broader Apertis project. hawkBit is already in use elsewhere, notably by Bosch as part of its IoT infrastructure[8], however more

[1] https://www.eclipse.org/hawkbit/
[2] https://jwd.pages.apertis.org/apertis-website/concepts/system-updates-and-rollback/
[3] https://jwd.pages.apertis.org/apertis-website/concepts/application-framework/#the-app-store
[4] https://hawkbit.apertis.org
[5] https://gitlab.apertis.org/infrastructure/apertis-image-recipes/-/pipelines
[6] https://gitlab.apertis.org/pkg/apertis-hawkbit-agent
[7] https://jwd.pages.apertis.org/apertis-website/guides/deployment-management/
[8] https://docs.bosch-iot-rollouts.com/documentation/index.html

work is required to reach the point where the Apertis infrastructure (or a deployment based on the Apertis infrastructure) would be ready for production use. In this document we will describe the steps we feel that need to be taken to provide a reference deployment that could be more readily suitable for production.

Evaluation Report

Server configuration

The current hawkBit deployment is hosted on Collabora's infrastructure. The example Docker Compose configuration file[9] has been modified to improve stability, security and adding a reverse proxy providing SSL encryption. This has been wrapped with Chef[10] configuration to improve maintainability. Whilst this configuration has limitations (that will be discussed later), it provides a better starting point for the deployment of a production system. These configuration files are currently stored in Collabora's private infrastructure repository and thus not visible to 3rd parties.

Considering the production workflow

The currently enabled process for the enrollment and configuration of a target device into the hawkBit deployment infrastructure requires the following steps:

- Install Apertis OSTree based image on the target device.
- Define or determine the `controllerid` for the device. This ID needs to be unique on the hawkBit instance as it is used to identify the target.
- Enroll the target on the hawkBit instance, either via the UI[11] or API[12].
  - If adding via the UI, hawkBit creates a security token, if adding via the API the security token can be generated outside of hawkBit.
- Modify the configuration file for `apertis-hawkbit-agent` to contain the correct URL for the hawkBit instance, the targets `controllerid` and the generated security token. This configuration file is `/etc/apertis-hawkbit-agent.ini`. Without these options being set, the target will be unable to find and access the deployment server to discover updates.

This workflow presents a number of points that could prove contentious in a production environment:

- A need for access to the hawkBit deployment server (that may be hosted on external cloud infrastructure) from the production environment to register the `controllerid` and security token.

[9] https://github.com/eclipse/hawkbit/blob/master/hawkbit-runtime/docker/docker-compose-stack.yml
[10] https://www.chef.io/
[11] https://www.eclipse.org/hawkbit/ui/#deployment-management
[12] https://www.eclipse.org/hawkbit/rest-api/targets-api-guide/#__post__rest__v1__targets

- The requirement to have a mechanism to add configuration to the device post software load.

The security token based mechanism is one of a number of options[13] available for authentication via the DDI API. The security token must be shared between the target and the hawkBit server. This approach has a number of downsides:

- The Token needs to added to the hawkBit server and tied to the target devices `controllerid`. This may necessitate a link between the production environment and an external network to access the hawkBit server.
- The need for the shared token to be registered with the server for authentication would make it impossible to use the "plug n' play" enrollment of the target devices supported by hawkBit.

hawkBit allows for a certificate based authentication mechanism (using a reverse proxy before the hawkBit server to perform authentication) which would remove the need to share a security token with the server. Utilizing signed keys would allow authentication to be achieved independently from enrollment, thus allowing enrollment to be carried out at a later date and would remove the need to store data per device in the hawkBit from the production environment. hawkBit allows for "plug'n play[14]" enrollment, the enrollment of the device when it's first seen by hawkBit, thus the device could potentially be enrolled once the end user has switched on the device and successfully connected it to a network for the first time when using certificate based authentication.

For many devices it would not be practical or desired to have remote access into the production firmware to add device specific configuration, such as a security token or device specific signed key. `apertis-hawkbit-agent` currently expects such configuration to be saved in `/etc/apertis-hawkbit-agent.ini`. An option that this presents is for the image programmed onto the target to provide 2 OSTree commits, one with the software expected on the device when shipped and the other for factory use, with boot defaulting to the latter. OSTree will attempt to merge any local changes made to the configuration when updating the image. The factory image could be used to perform any testing and factory configuration tasks required before switching the device to the shipping software load. Customizations to the configuration made in the factory should then be merged as part of the switch to the shipping load, and the factory commit can be removed from the device. Such an approach could provide some remote access to the target as part of the factory commit, but not the shipping commit, thus avoiding remote access being present in the field.

As previously mentioned, a unique `controllerid` is needed by hawkBit to identify the device and needs to be stored in the configuration file. An alternative approach may be to generated this ID from other unique data provided by the device, such as a MAC address or unique ID provided by the SoC used in the device.

[13] https://www.eclipse.org/hawkbit/concepts/authentication/
[14] https://gitter.im/eclipse/hawkbit/archives/2016/07/27

Management UI access

We currently have a number of static users defined with passwords available to trusted maintainers. Such as scheme is not going to scale in a production environment, nor provide an adequate level of security for a production deployment. hawkBit provides the ability to configure authentication using a provider implementing the OpenID Connect standard, which would allow for much greater flexibility in authenticating users.

Enabling device filtering

hawkBit provides functionality to perform update rollouts in a controlled way, allowing a subset of the deployed base to get an update and only moving on to more devices when a target percentage of devices have received the update and with a configurable error rate. When rolling out updates, in an environment where more than one hardware platform or revision of hardware is present, it will be necessary to be able to ensure the correct updates are targeted towards the correct devices. For example, two revisions of a gadget could use different SoCs with different architectures each requiring a different build of the update and different versions of a device may need to be updated with different streams of updates. In order to cater for such scenarios, it is important for hawkBit to be able to accurately distinguish between differing hardware. Support to achieve this is provided via hawkBit's ability to store attributes. These attributes can be set by the target device via the DDI interface once enrolled and used by hawkBit to filter target devices into groups. At the moment the `apertis-hawkbit-agent` is not setting any attributes.

Provisioning for multiple product teams or partners

In order to use hawkBit for multiple products or partners it would be either beneficial or necessary for each to have some isolation from each other. This could be achieved via hawkBit's multi-tenant functionality or via the deployment of multiple instances of hawkBit. It is likely that both of these options would be deployed depending on the demands and requirements of the product team or partner. It is expected that some partners may like to use a deployment server provided by Apertis or one of it's partners. In this instance multi-tenancy would make sense. Others may wish to have their own instance, possibly hosted by themselves, in which case providing a simple way to deploy a hawkBit instance would be beneficial.

Deploying multiple instances of hawkBit using the docker configuration would be trivial. The multi-tenant configuration requires the authentication mechanism for accessing the management API, web interface and potentially DDI API to be multi-tenant aware.

Life management of artifacts

The GitLab CI pipeline generally performs at least 2 builds a day, pushing multiple artifacts for each architecture and version of Apertis. In order to minimize the space used to store artifacts and so as not to store many defunct artifacts, they are currently deleted after 7 days.

Whilst this approach enables the Apertis project to frequently exercise the artifact upload path and has been adequate for Apertis during it's initial phase, a more comprehensive strategy will be required for production use. For shipped hardware, it is unlikely that any units will be updated as frequently. In addition, depending on the form and function of the device, it may only poll the infrastructure to check for updates sporadically, either due to the device not needing to be on or not having access to a network connection capable of reaching the deployment server. Artifacts will needed to be more selectively kept to ensure that the most up-to-date version is kept available for each device type and hardware revision. Older artifacts that are no longer the recommended version should be safe to delete from hawkBit as no targets should be attempting to update to them.

Platform scalability

hawkBit provides support for clustering to scale beyond the bandwidth that a single deployment server could handle. The Apertis hawkBit instance is not expected to need to handle a high level of use, though this may be important to product teams who might quite quickly have many devices connected to hawkBit in the field.

Recommendation

Server Configuration

- The improvements made to the Docker Compose configuration file should be published either in a publicly visible Apertis repository and/or improvements should be submitted back to the hawkBit project to be included in the reference Docker configuration.

Considering the production workflow

- The hawkBit deployment should be updated to use a signed key based security strategy.
- `apertis-hawkbit-agent` should be improved to enable authentication via signed keys.
- `apertis-hawkbit-agent` should be improved to auto-enroll when the target device is not already found.

- `apertis-hawkbit-agent` is currently storing its configuration in `/etc`, this should be extended to look under `/var` and the default configuration should be moved there.
- A mechanism should be added to `apertis-hawkbit-agent` to enable the `controllerid` to be generated from supported hardware sources.

Management UI access

- The Apertis hawkBit instance should be configured to use the OpenID authentication mechanism, ideally using the same SSO used to authenticate users for other Apertis resources.

Enabling device filtering

- Update `apertis-hawkbit-agent` to set attributes based on information known about the target device. This should include (where possible):
  - Device Architecture
  - Device Type
  - Device Revision

Provisioning for multiple product teams or partners

- Apertis does not have a direct need for a multi-tenant deployment nor for multiple deployments. Investigate and document what's involved for setting up a multi-tenanted installation.

Life management of artifacts

- Apertis is developing a base platform to be used by production teams and thus the images it produces for it's reference hardware needs a subtly different scheme[15] from that which would be anticipated to be needed by a production team. It is therefore recommended that the process removing old artifacts should adhere to the following rules:
  - Retain all point releases for current Apertis releases
  - Retain 7 days of daily development builds
  - Delete all artifacts for versions of Apertis no longer supported

Platform scalability

- At this current point in time we do not feel that investigating platform scalability has immediate value.

[15] https://jwd.pages.apertis.org/apertis-website/architecture/long-term-reproducibility/