Designing a Secure IoT System Architecture from a Virtual Premise for a Collaborative AI Lab

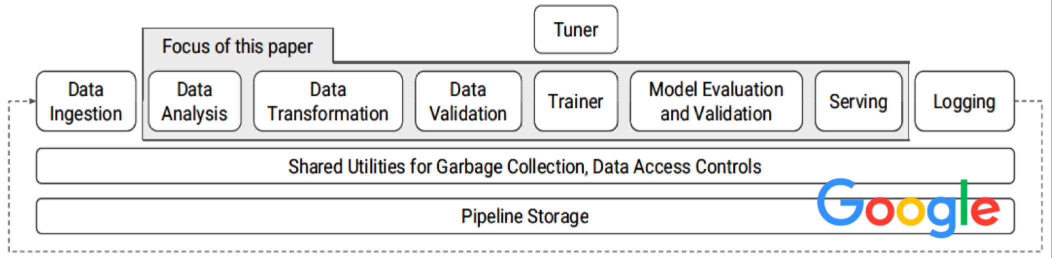
Vida Ahmadi Mehri
Dragos Ilie
Kurt Tutschku

Blekinge Institute of Technology (BTH), Sweden

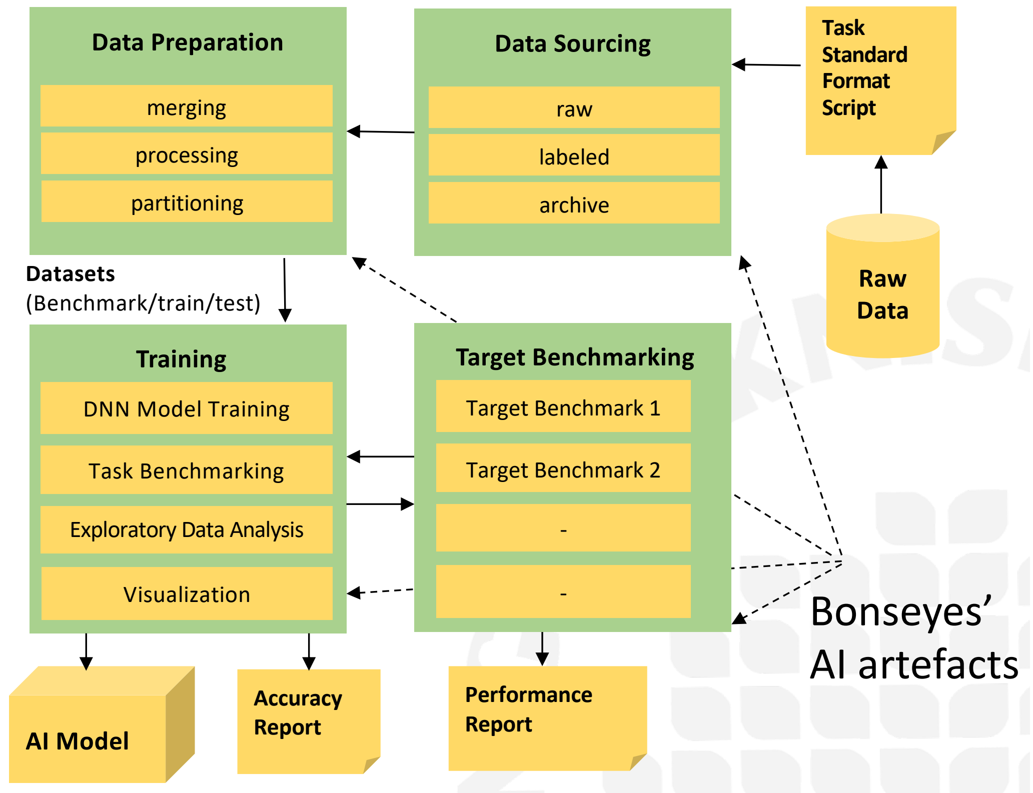
Outline

- Background - AI pipelines
- Bonseyes project
- Bonseyes architecture
- Authentication and access management
- Lessons learned
- Conclusion and future work

AI pipeline and AI artefacts



From D. Baylor *et.al.*, *"TFX: A tensorflow-based production-scale machine learning platform."*, in 23rd ACM SIGKDD, 2017

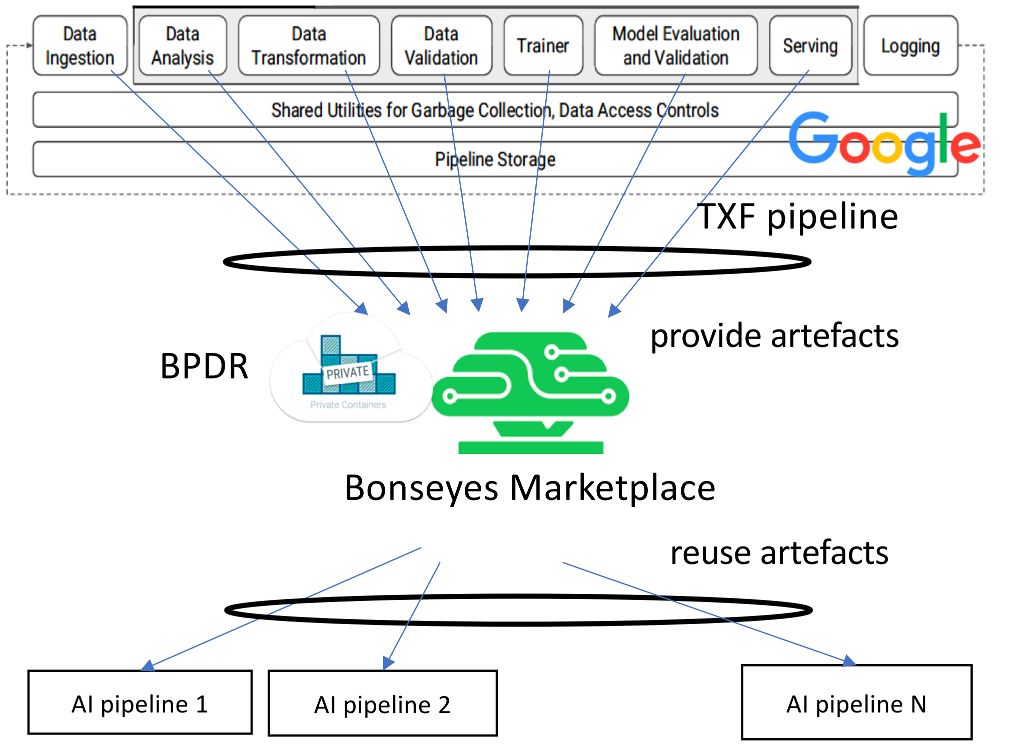


Bonseyes project

- How are AI pipelines related to IoT system architecture?
- Bonseyes problem statement[1]:
  - ➢AI is rapidly spreading to embedded and IoT devices
  - ➢Optimized end-to-end systems for deep learning require the integration of data, algorithms, tools and dedicated hardware
  - ➢Only a few large companies (Google, Apple) are able to build such systems
    - ❑Partly due to having privileged access to data (data-wall problem)
- Bonseyes approach:
  - ➢Define an open architecture to enable an eco-system of companies to collaborate in building complex distributed systems
  - ➢Design an AI pipeline framework to provide key benefits such as scalability, reusability, reproducibility and flexibility.

[1]dePrado *et.al.*, "*AI Pipeline - bringing AI to you. End-to-end integration of data, algorithms and deployment tools*", arXiv.org, 2019.

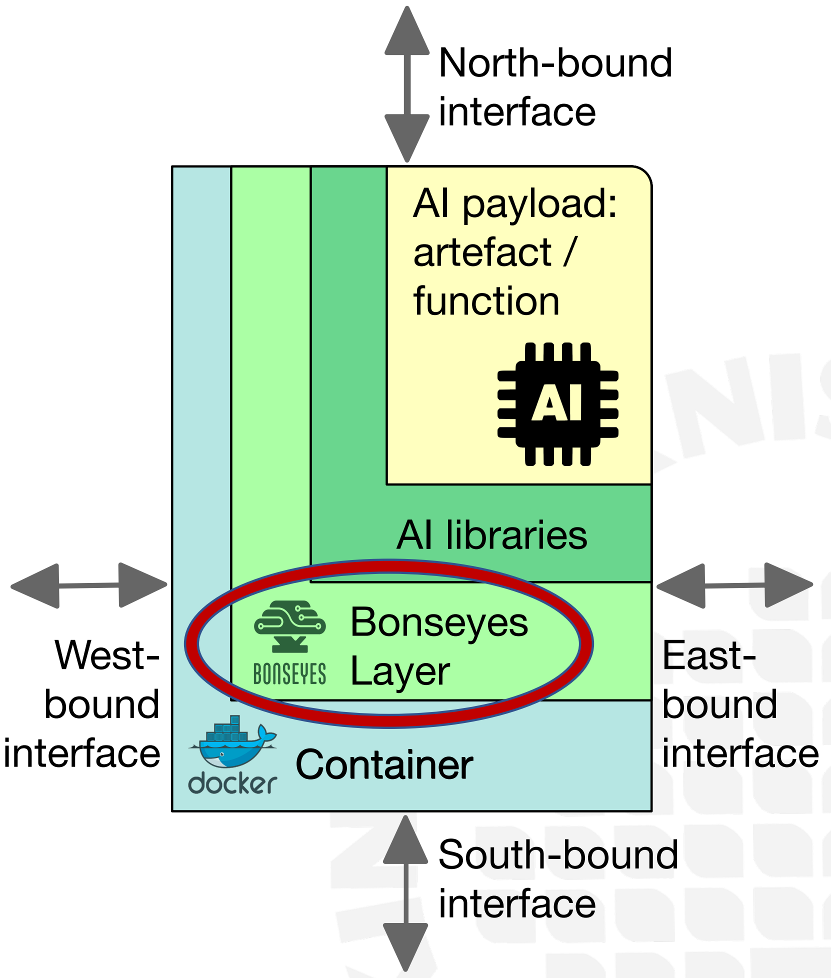
AI marketplace



Derived from D. Baylor *et.al.*, *"TFX: A tensorflow-based production-scale machine learning platform."*, in 23rd ACM SIGKDD, 2017

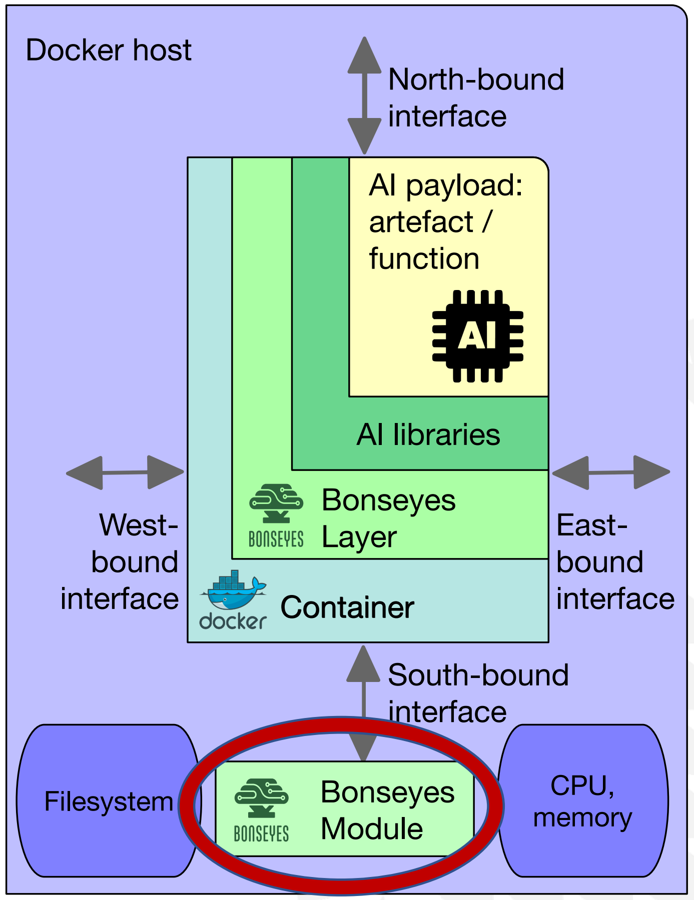
AI artefact

- AI payload:
  - ➢ Raw data or features
  - ➢ Interface towards data lakes
  - ➢ AI algorithms/functions
- AI libraries
  - ➢ AI frameworks
  - ➢ System libraries
- Bonseyes layer (BL)
  - ➢ Manage incoming/outgoing artefact communication
  - ➢ Enable secure APIs to access AI functions and content
  - ➢ Facilitate artefact authentication with other entities from the Bonseyes eco-system
- Container: Docker



AI artefact on Docker host

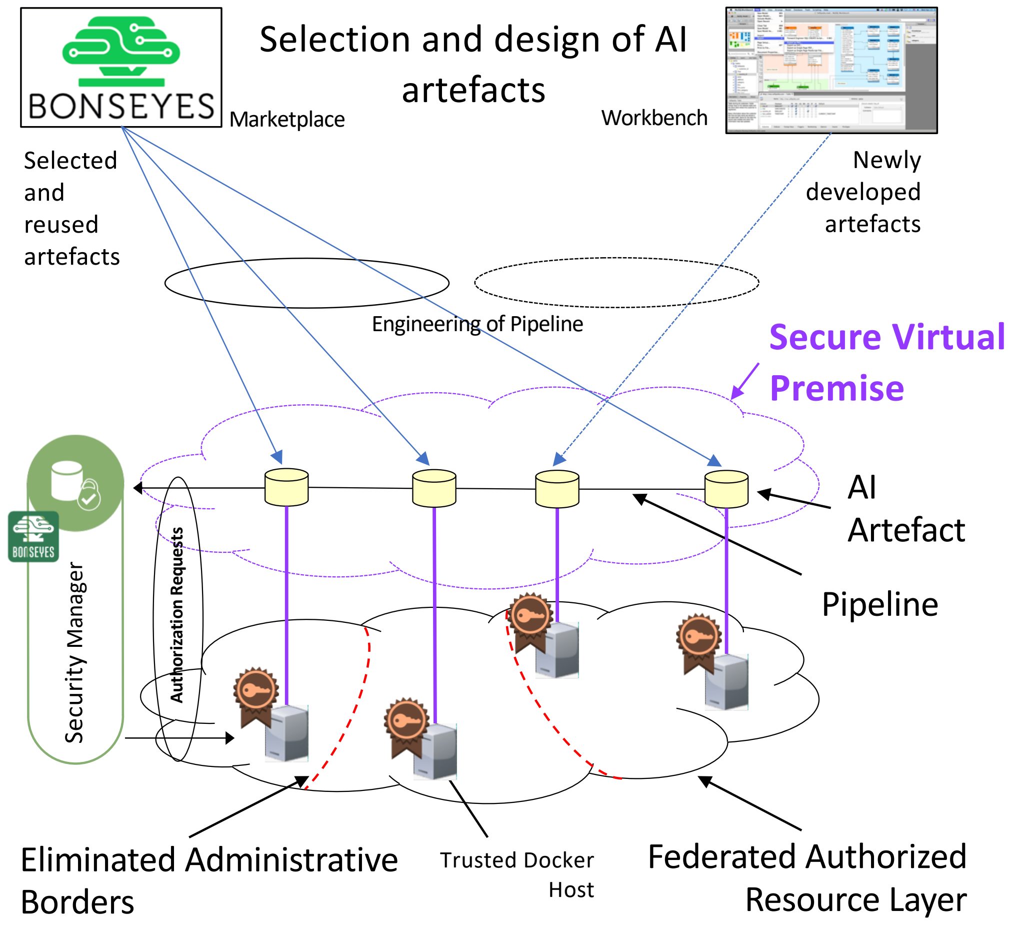
- Bonseyes module (BM) facilitates authentication of the host
  - ➢Plays similar role for host as the Bonseyes layer (BL) does for the artefact.
- BM and BL must mutually authenticate and authorize before the artefact is allowed to execute



An agile approach to security

- Distributed pipelines due to scalability concerns
- Top security concerns (unordered):
  - *Licensed-controlled artefact usage*
    - Attribute-based access control
  - **Confidentiality and integrity for pipeline communication**
  - **Trustworthy collaborative environment**
  - Compliance with privacy laws (e.g., GDPR)
- Aim: provide a *virtual premise* where only authenticated entities are allowed to engage in collaboration
- Modify system gradually to address security issues

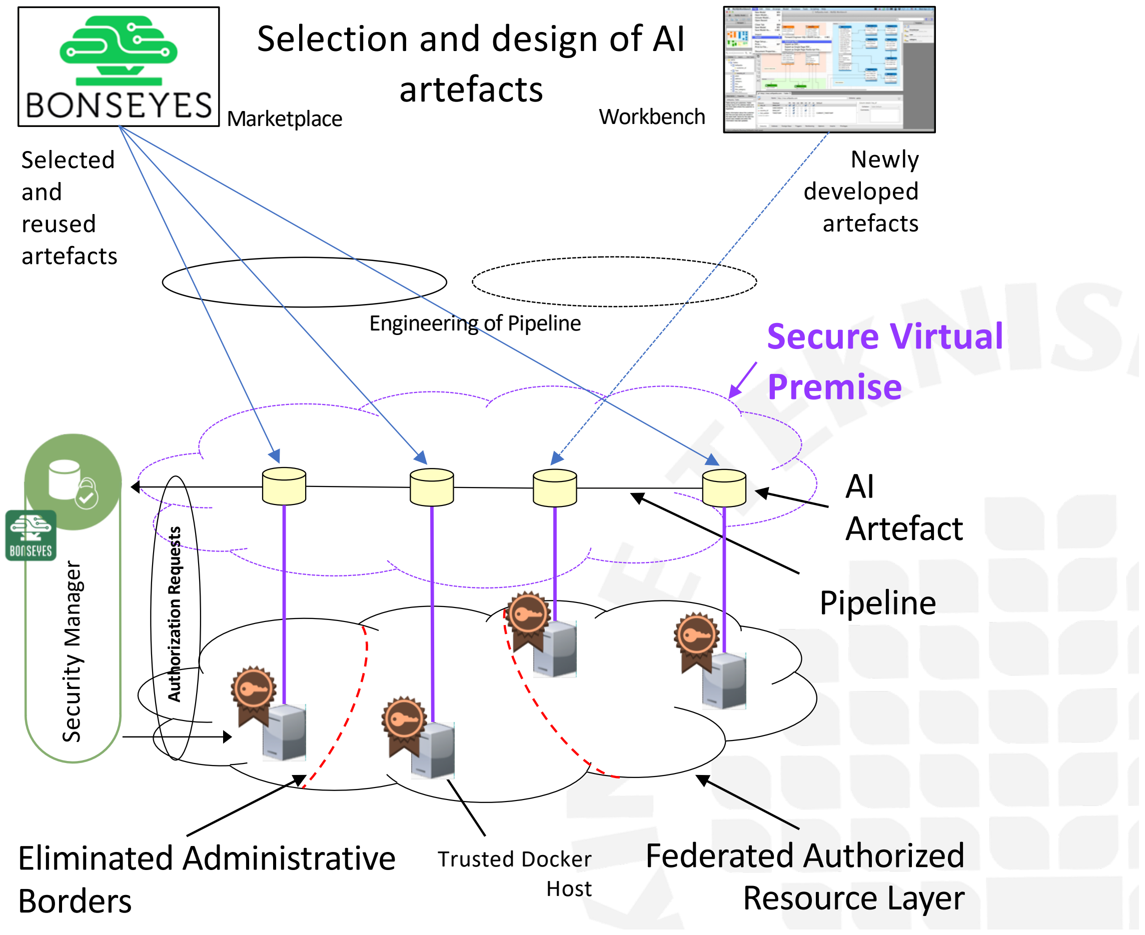
Proposed architecture



BONSEYES
Marketplace
Selection and design of AI artefacts
Workbench
Selected and reused artefacts
Newly developed artefacts
Engineering of Pipeline
Secure Virtual Premise
AI Artefact
Pipeline
Security Manager
Authorization Requests
Eliminated Administrative Borders
Trusted Docker Host
Federated Authorized Resource Layer

Actors and roles

Actor	Role
Bonseyes layer (BL)	Located inside the Docker image. Manages the authenticity of the artefact.
Bonseyes module (BM)	Located on the Docker host. Manages the authenticity of the host.
Virtual Premise (VP)	A set of Bonseyes registered hosts (each having a valid BM). Used for executing one or more pipelines
Marketplace (MP)	Meeting place for Bonseyes users. Contains the docker images and the SM.
Security Manager (SM)	Located inside the MP. Manages identities and crypto material. Contains a certification authority (CA). Trust anchor.
User	AI developer creating and executing pipelines.



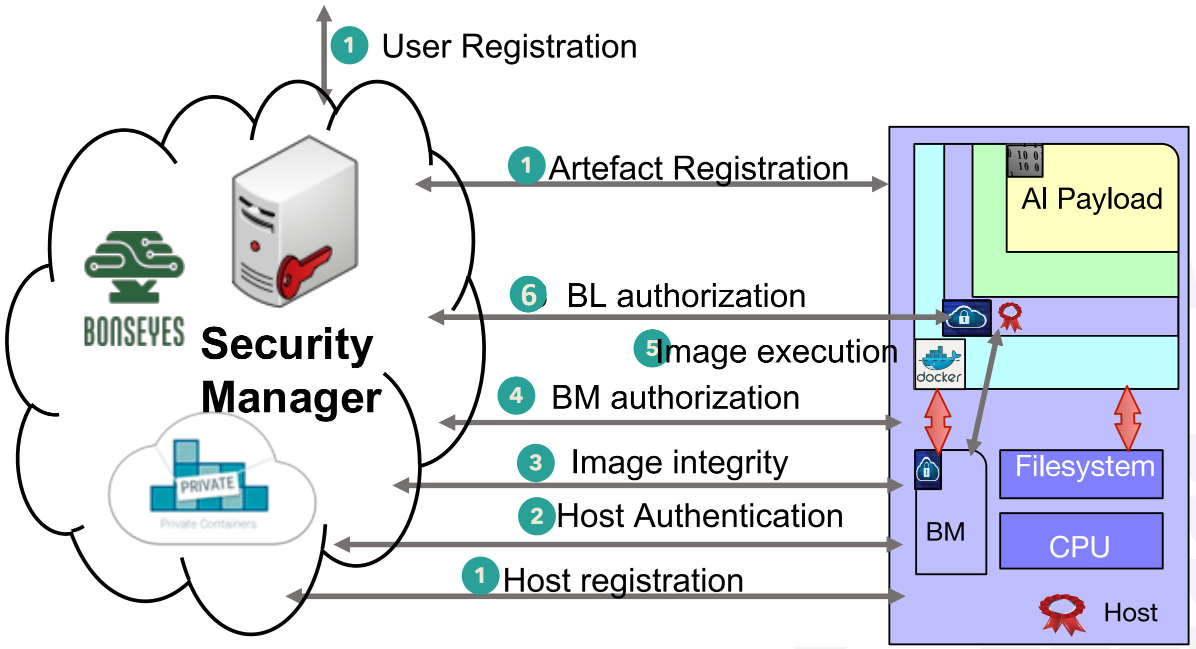
BONSEYES
Marketplace
Selection and design of AI artefacts
Workbench
Selected and reused artefacts
Newly developed artefacts
Engineering of Pipeline
Secure Virtual Premise
AI Artefact
Pipeline
Security Manager
Authorization Requests
Eliminated Administrative Borders
Trusted Docker Host
Federated Authorized Resource Layer

Authentication and access management

- PKI-based approach
- Simplifying assumptions
  - SM is not compromised
  - Federated administrative domains are not malicious, nor do they collude with adversaries
  - Docker images uploaded to MP do not contain malware, including backdoors
  - Entities are capable of communicating over HTTPS using certificate-based authentication (uWSGI server-side, Python Requests client-side)
- Goal: automatically authenticate peers and install cryptographic material required by HTTPS on authorized hosts and artefacts.
  - Enables certificate-based authentication for TLS clients and servers

Authentication and access management

1. Artefact, host, and user registration
2. Host authentication
3. Image integrity check
4. Host (BM) authorization for artefact execution
5. Image execution (BL start)
6. Artefact (BL) authorization for execution on host.



1. Artefact, host, and user registration

- Artefact registration
  - User uploads base Docker image to MP and configures policies regarding its execution
  - Image is verified for malicious content before being made available to other users
  - BL is added to the base image
  - A unique human-readable identifier is inserted into the image
    - Signed with SM private key (*artefactID*)
  - SM's root and intermediate certificates are inserted into the image
  - SM computes a message digest over the final image
  - SM signs the digest creating the *image ID*.
  - Image ID and artefact ID are stored in MP
- Host registration
  - Hardware ID is computed and tied to the host owner's identity by digital signature (*host ID)*
  - Host owner registers the hardware ID with MP and configure policies regarding its use (*e.g.,* which VP it belongs to)
  - Owner installs BM (containing SM certificates and host-specific asymmetric key pair and certificate) on the host
- User registration
  - Unique *user ID*
  - Digital user certificate (PKCS#12/PFX format) installed in the web browser and/or AI workbench

2. Host authentication (BM start)

- BM is started on the host after OS boots up
- BM opens HTTPS connection to the SM and identifies itself with the host ID
- uWSGI / Flask inside BM is configured to use the cryptographic material for HTTPS and client authentication
- SM marks the host as available resource

3. Image integrity check

- User requests SM to assist in setting up an AI pipeline on a specific VP.
- If request granted:
  - ➢SM connects to each BM (host) belonging to the pipeline and instructs it to download the Docker images
  - ➢BM computes message digest for each retrieved image and sends it to SM for verification
  - ➢If verification fails, pipeline construction is aborted and BMs are instructed to remove the images from local storage

4. Host authorization for artefact execution

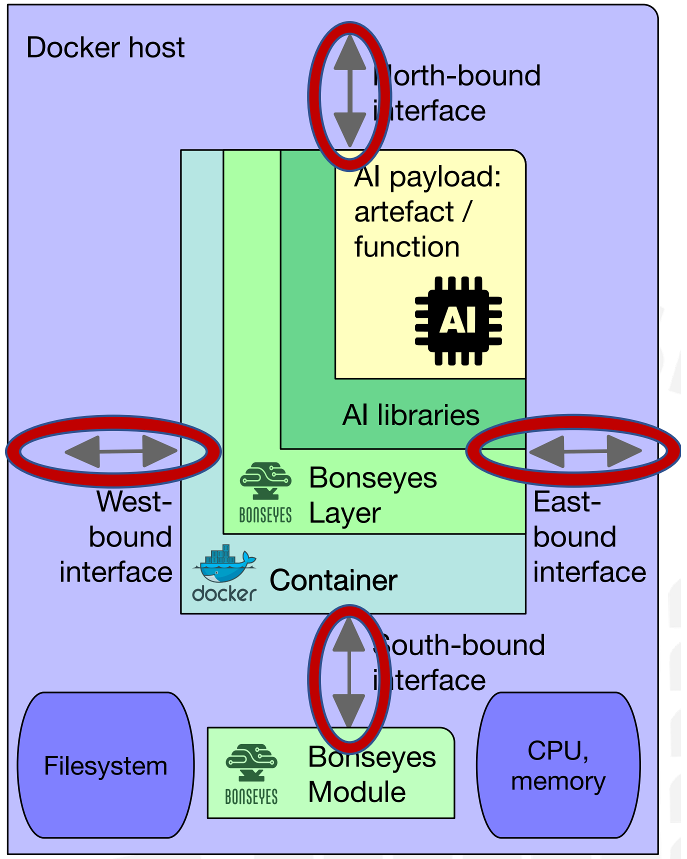
- BM requests from the SM execution policies for the retrieved artefacts
- The request contains:
  - Message digest of Docker image
  - Host ID
  - User ID
  - Nonce (to identify a specific instance of the artefact)
- Request is sent over HTTPS
- SM looks up the host policy specification and use it to constructs a *host license* signed with its private key. Contains among other things:
  - The *runtime ID*: (image digest, artefact ID, user ID, host ID, host nonce, host IP address)
  - Asymmetric key pair associated with runtime ID and corresponding certificate
- Host license is returned to invoking BM.

5. Image execution (BL start)

- If a valid host license is obtained, BM starts the Docker image, which in turns start BL
- Host license is passed to BL upon start
  - BL uses SM certificate to verify the license signature (indication it is executed on a genuine host)
  - Upon successful verification, the authentication process moves to the final step.

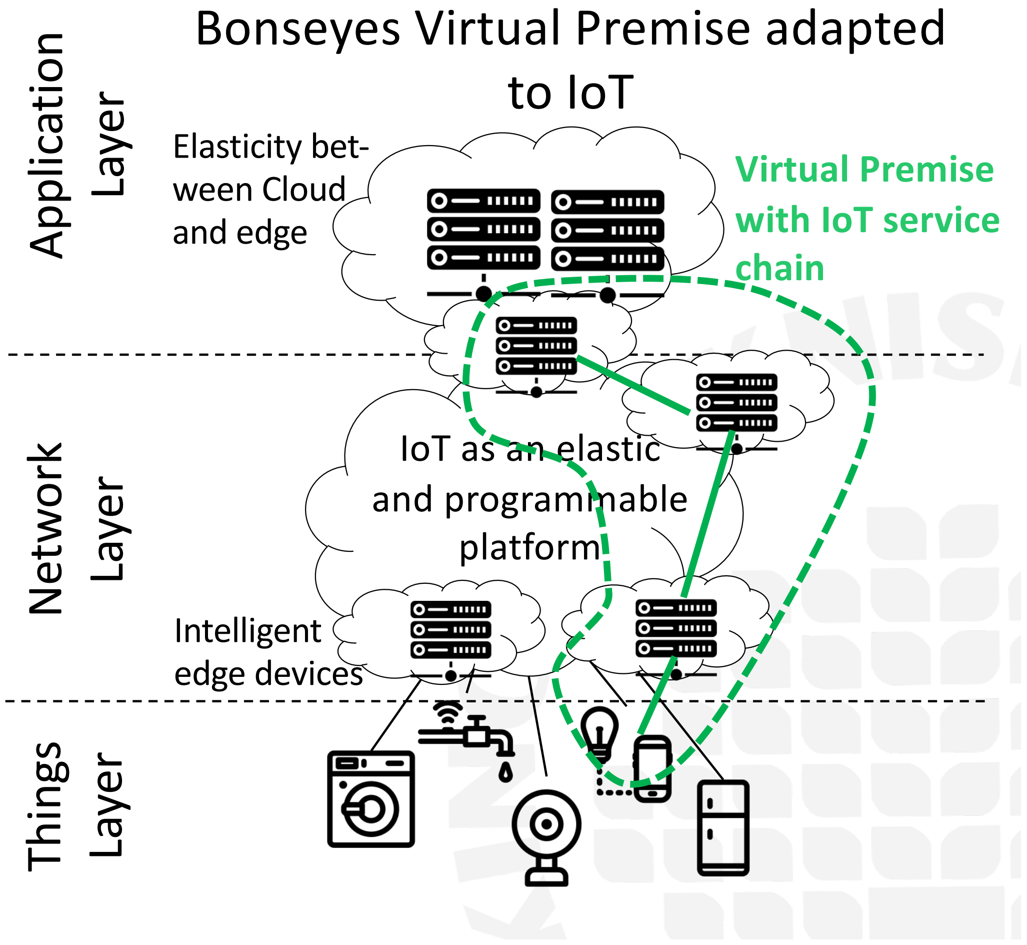
6. Artefact authorization for execution on host

- BL requests from the SM execution policies specific to the host where it runs
- The request contains:
  - ➢ Received host license
  - ➢ Own artefact ID
- Request is sent over HTTPS
- SM looks up the artefact policy specification and use it to constructs an artefact *license* signed with its private key.
- Artefact license is returned to BL
- uWSGI / Flask inside BM is configured to use the cryptographic material for HTTPS and client authentication
- BL forwards the artefact license to BM.
- BL-BM mutual authentication is complete, AI pipeline is setup and can begin execution.



Generalization to other IoT systems - Lessons learned -

- IoT service chains require security and compliance verification and enforcement
  - Along the workflow (horizontally): authentication for cooperation
    - Enables collaborative and distributed workflows
  - Vertically: authentication for resources
    - Enables code mobility
- Trustworthy environments based on multi-tenant infrastructure, containers of different origins and 3rd party users requires mutual authentication between physical and virtual components
  - BM-BL model can be applied here
- Docker technology
  - Flexible, portable
  - Difficult to implement strong security


Bonseyes Virtual Premise adapted to IoT
Application Layer
Network Layer
Things Layer
Elasticity between Cloud and edge
Virtual Premise with IoT service chain
IoT as an elastic and programmable platform
Intelligent edge devices

Current challenges and future work

- SM as a single point of failure
  - Distributed SM
- Weak host authentication
  - Relies on owner's honesty and assumptions of non-collusion
  - Remote attestation with TPM/TXT
- Pending evaluation
  - Integration with nViso AI pipeline
  - Icelandic OpenStack provider
    - HPCaaS, GPU support
  - Limitation: Docker containers within KVM VMs

Thank you for your attention!