



***Ground and Space Development,
Security, and Operations for
Reconfigurable Edge Platforms***

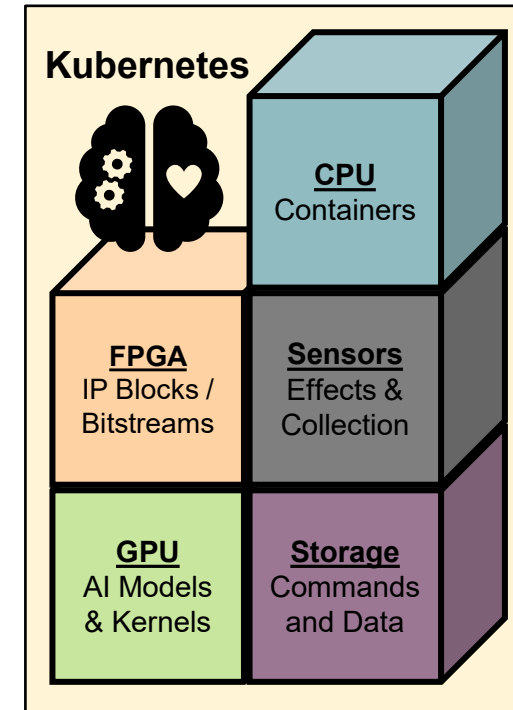
***Shariar Alamgir, Elisabeth Nguyen,
Dhruv Bohra
The Aerospace Corporation***

February 27, 2024

Transitioning Toward Dynamic & Distributed Space Cloud Architectures

Then vs Now

- Satellites today
 - Are programmed by a **selected** contractor
 - Can only rely on **themselves**
 - Require **squads of ground operators** for safe flight
 - Can only be used by a **few people**
 - Need to be told **exactly** what to do all the time
 - Largely **single-purpose** after launch
- Future satellites need to
 - Be programmable by **anyone**
 - **Cooperate** with other satellites
 - Operate with **less reliance** on humans
 - Receive the tasks **best** for them to execute
 - Autonomously determine **how** to fulfill requests
 - **Take over** for satellites suddenly out of commission



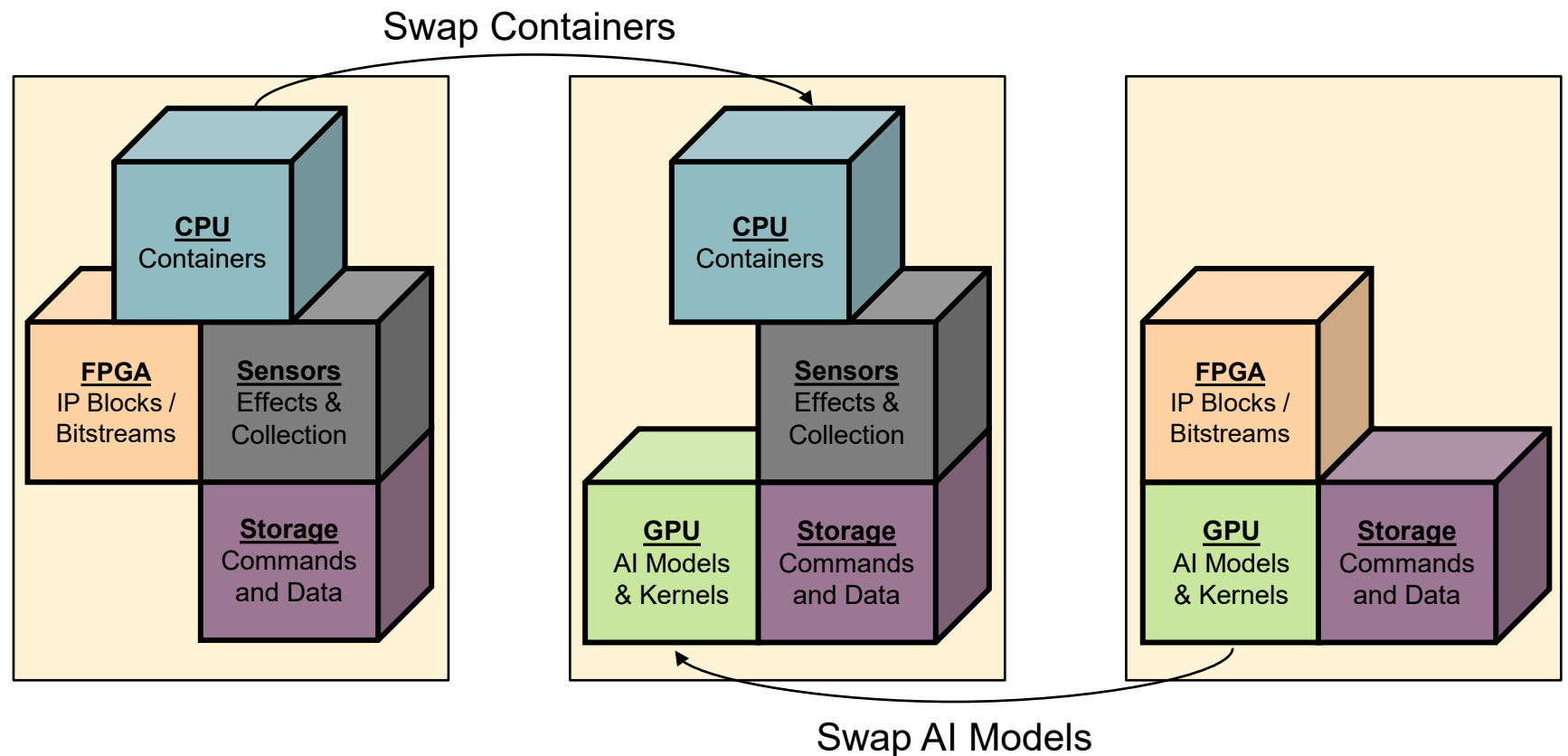
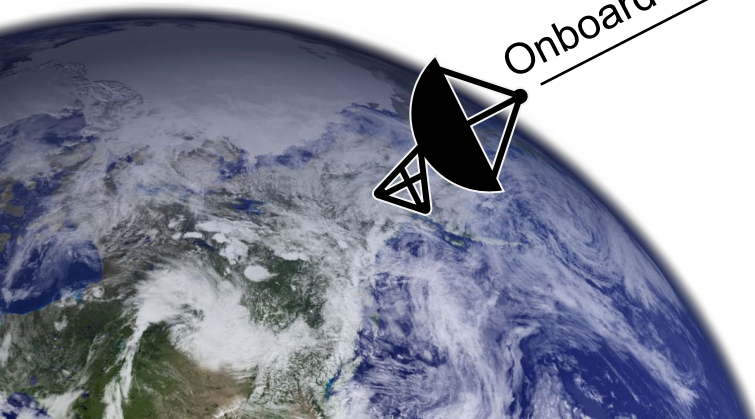
Ground systems are heavily responsible for providing reconfigurability for complex compute



Future Dynamic & Distributed Space Cloud Architecture

New ground challenges

- Securely create and validate the interfaces between modules targeting various compute elements
- Leverage autonomous pipelines to accelerate FSW V&V from years to days
- Commit support to legacy and future combinations of complex mission payload configurations

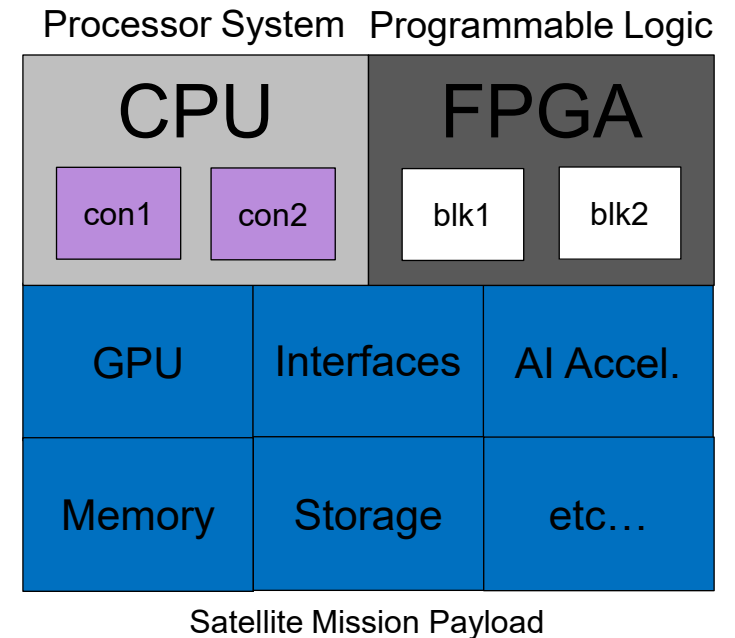




Example Scenario

Enabling Technology: Heterogeneous Compute

1. Overutilized satellite contains multiple compute types
 - Field-Programmable Gate Array with partial bitstream reconfiguration
 - GPU or AI accelerator capable of model segmentation / paravirtualization
 - General Purpose CPU with runtime replacement support
2. Incoming task requires new app, with all new onboard modules
 1. New FPGA blocks: updated sensor interfacing algorithm
 2. New GPU kernel: updated mission data preprocessor
 3. New CPU container: updated downlink target and SDR control
3. Requires an agile DevSecOps pipeline that
 - can quickly develop safe modules for all compute types
 - handle packaging of payload
 - verify operation in space-like environment

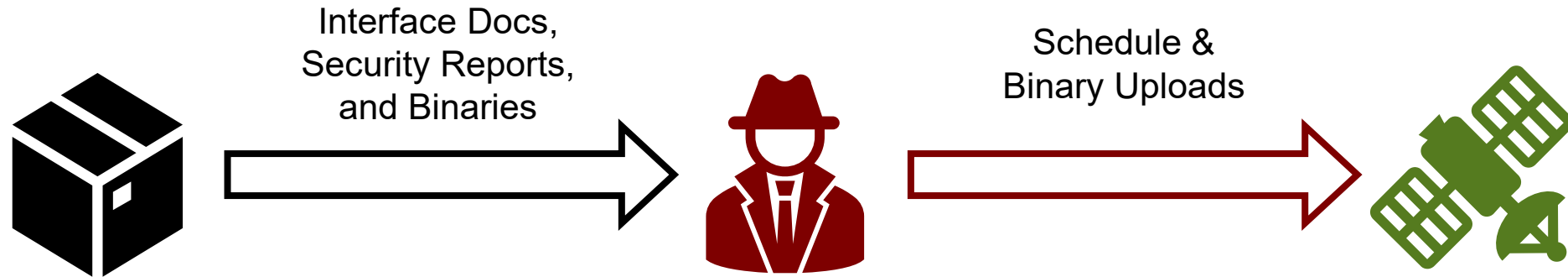


Future DevSecOps tooling will require pipeline standardization for compute elements apart from CPU containers



DevSecOps Sequence of Events

Reference Architecture Overview



Developer

- **Develop** source code
- Generate **Security** reports
- Test in **Operational** dev sandbox

Integrator

- **Develop** interface control document (ICD) and infrastructure SW
- **Secure** interfaces and verify reports
- **Operational** end-to-end tests

Each phase of the DevSecOps pipeline will be affected by the need to support space clouds

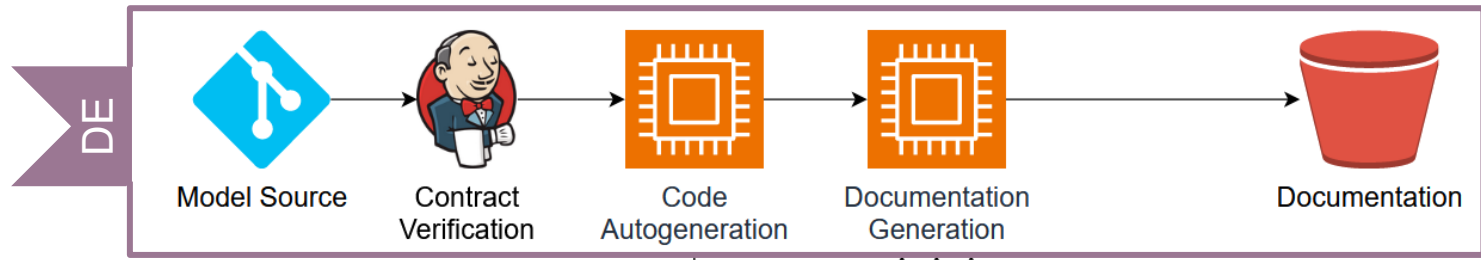


Developer Pipeline

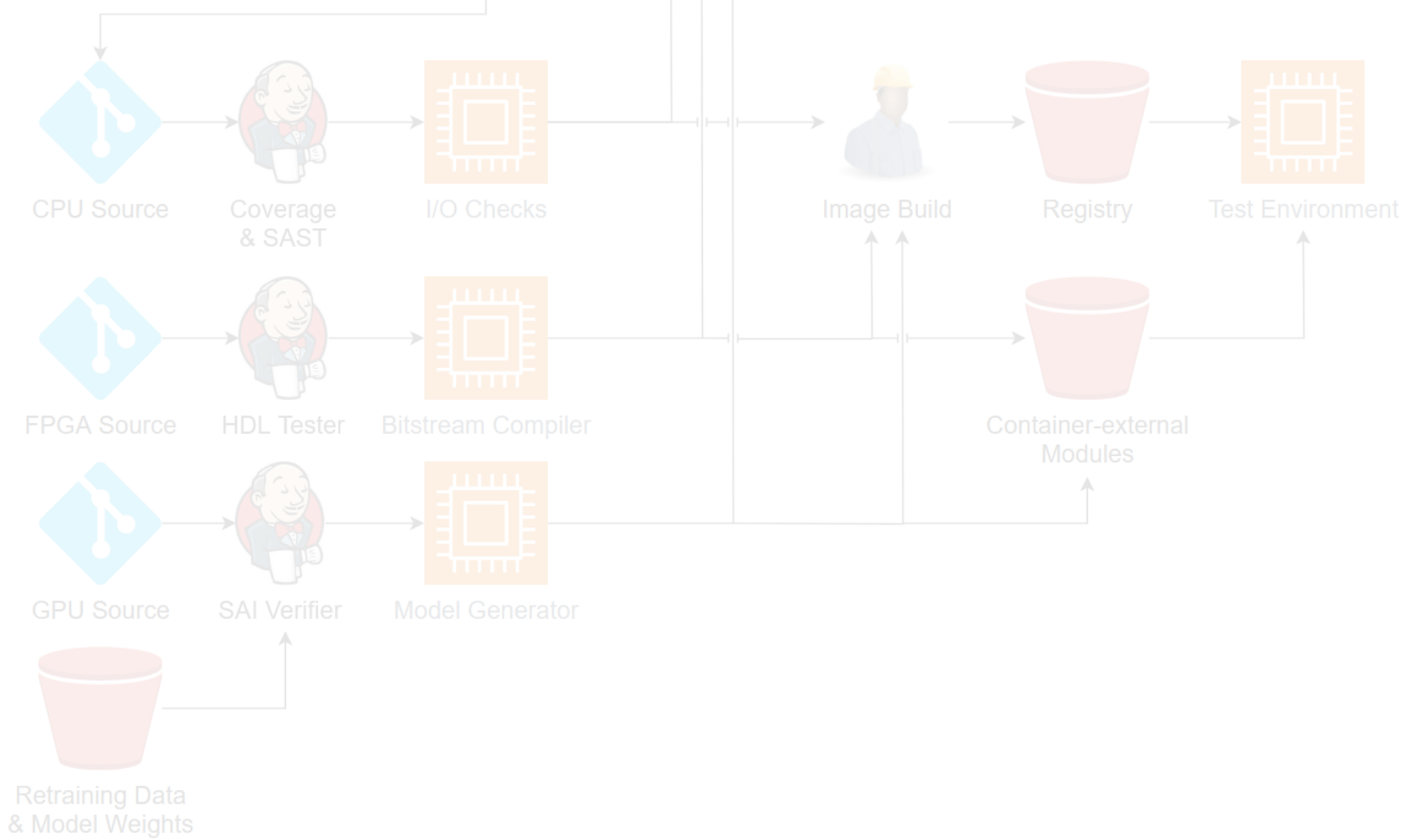


Reference Architecture

Developer Pipeline



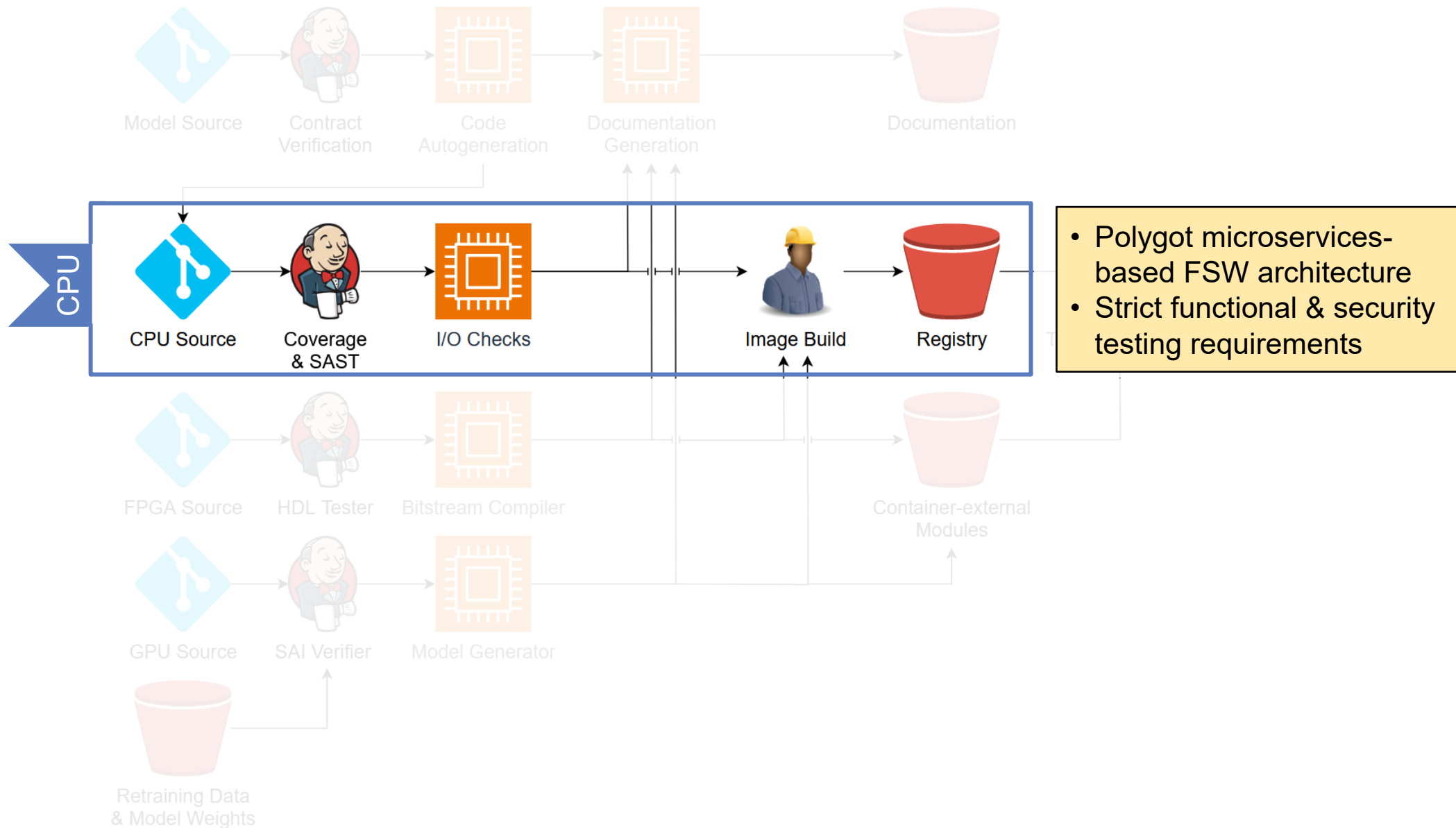
- Code autogeneration “guarantees” functionality matches API documentation
- Expressive models can be reused in anomaly resolution process





Reference Architecture

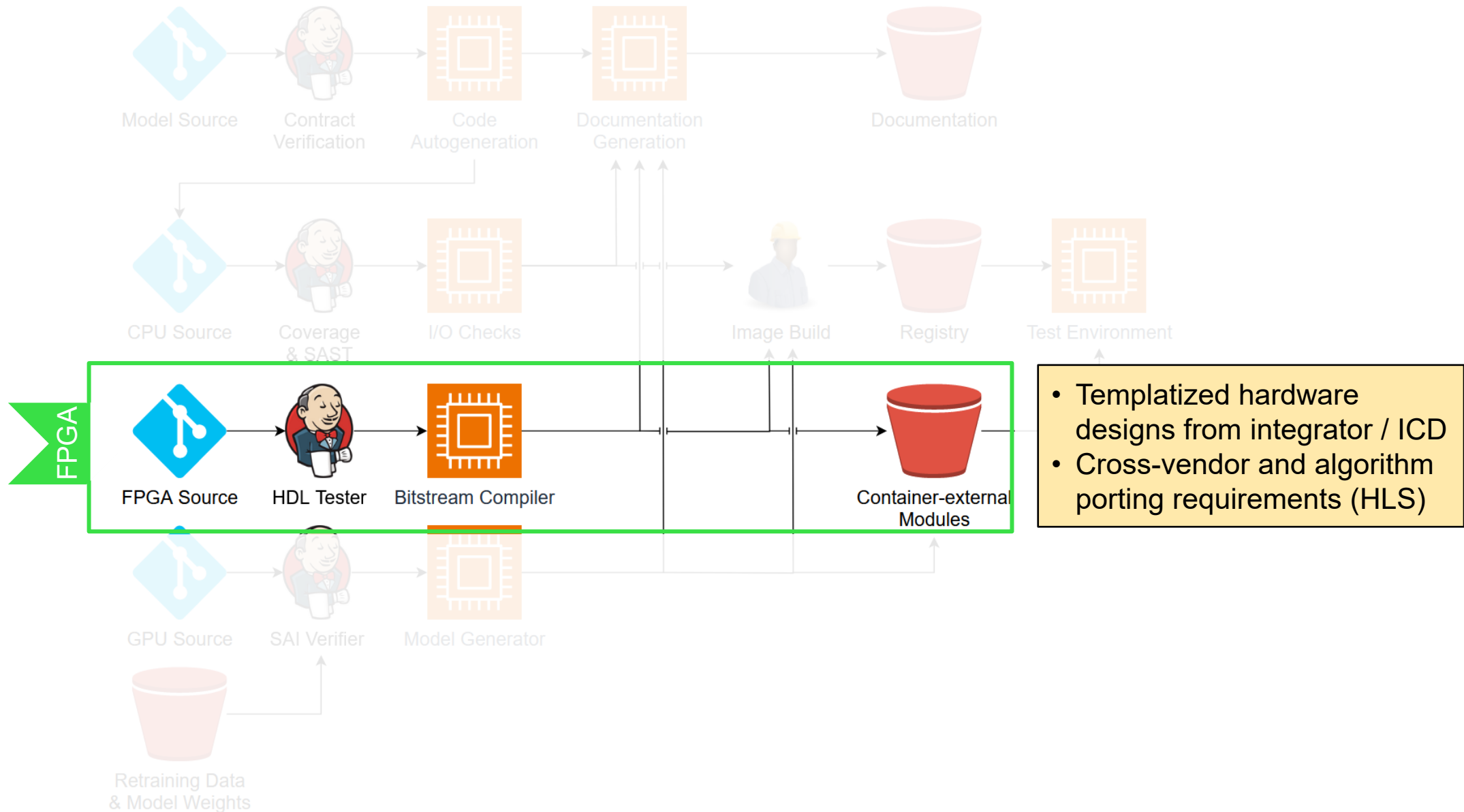
Developer Pipeline





Reference Architecture

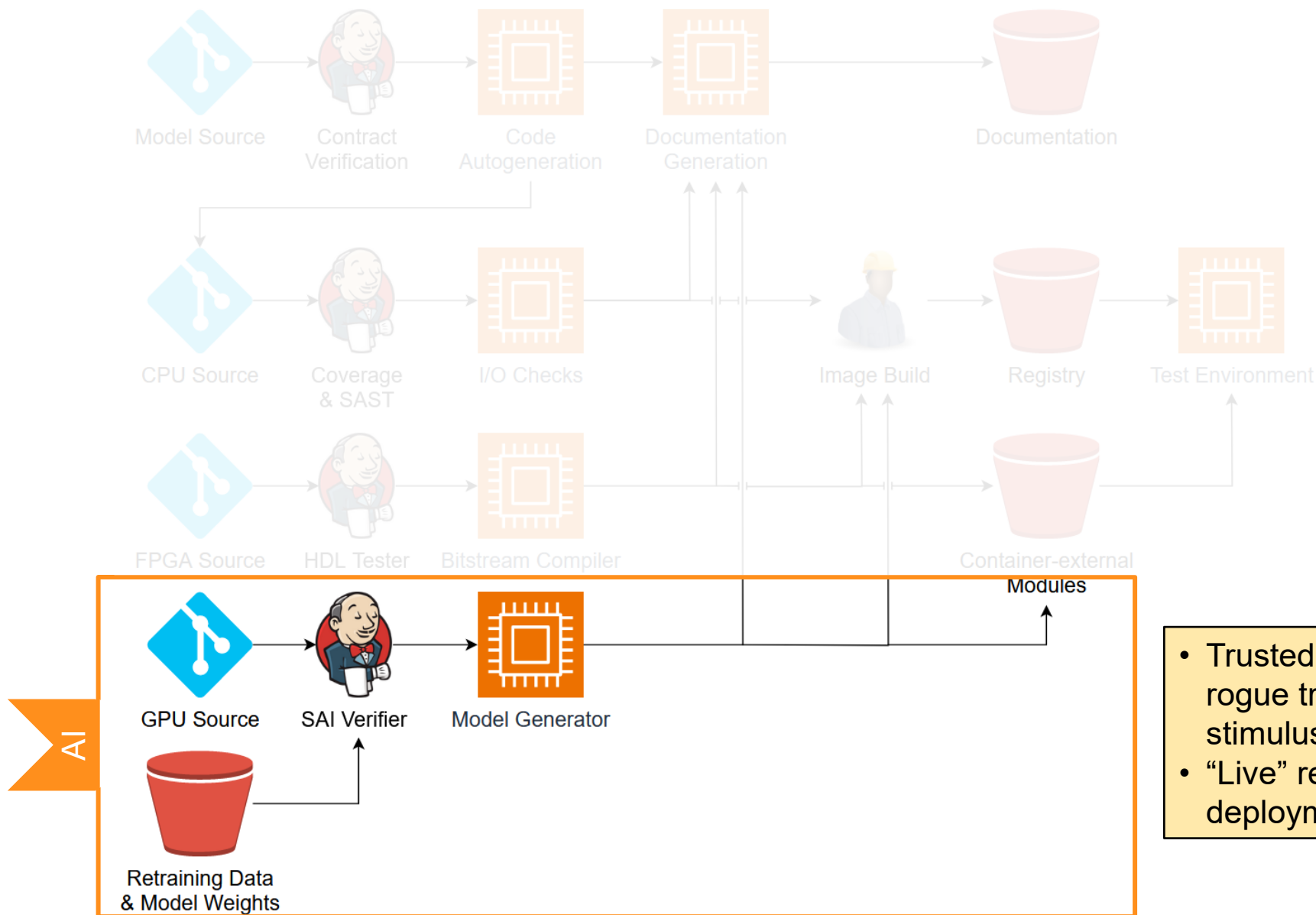
Developer Pipeline





Reference Architecture

Developer Pipeline



- Trusted AI testing against rogue training, adversarial stimulus, etc.
- “Live” retraining and deployment of networks



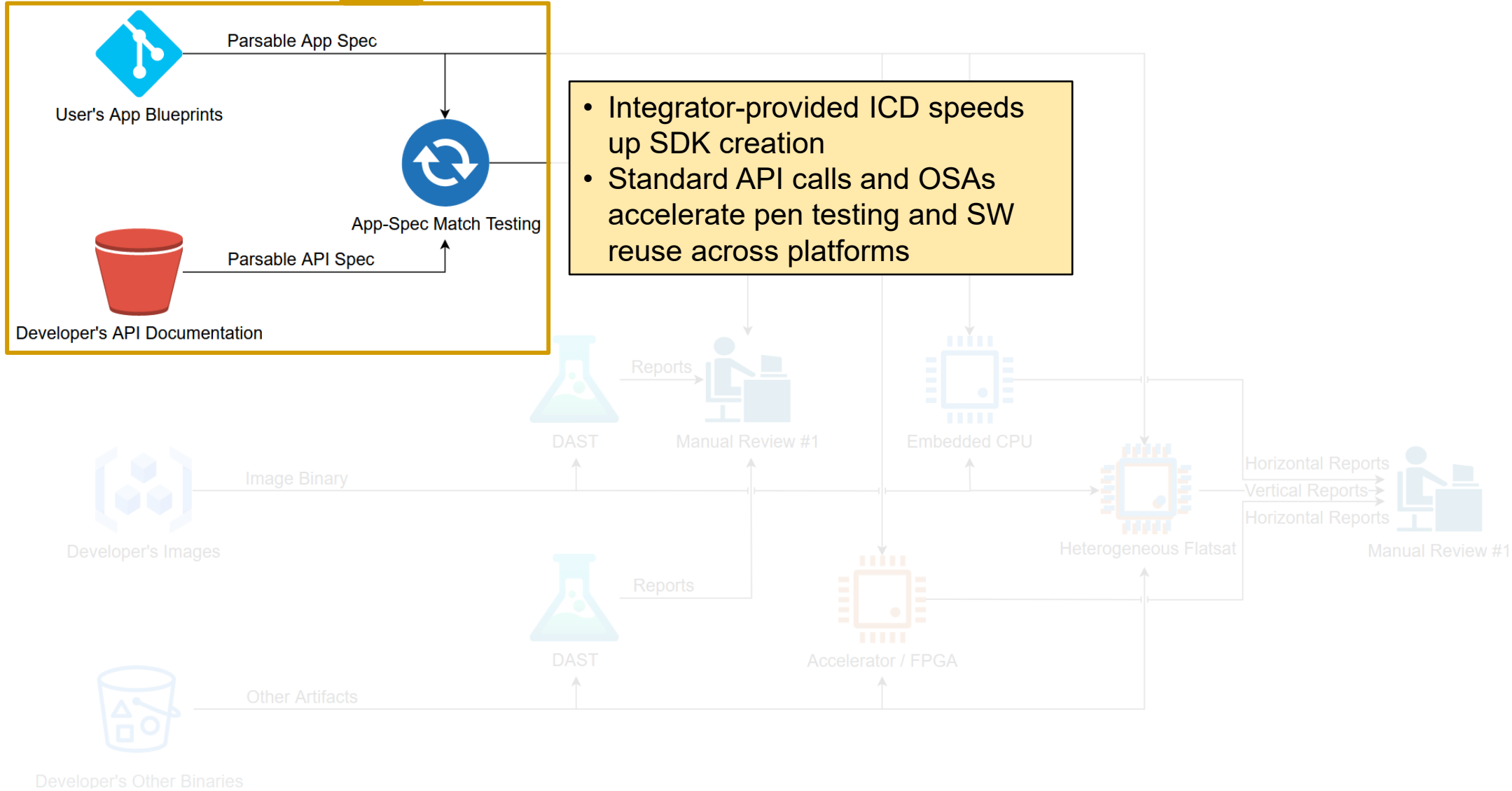
Integrator Pipeline



Reference Architecture

Integration Pipeline

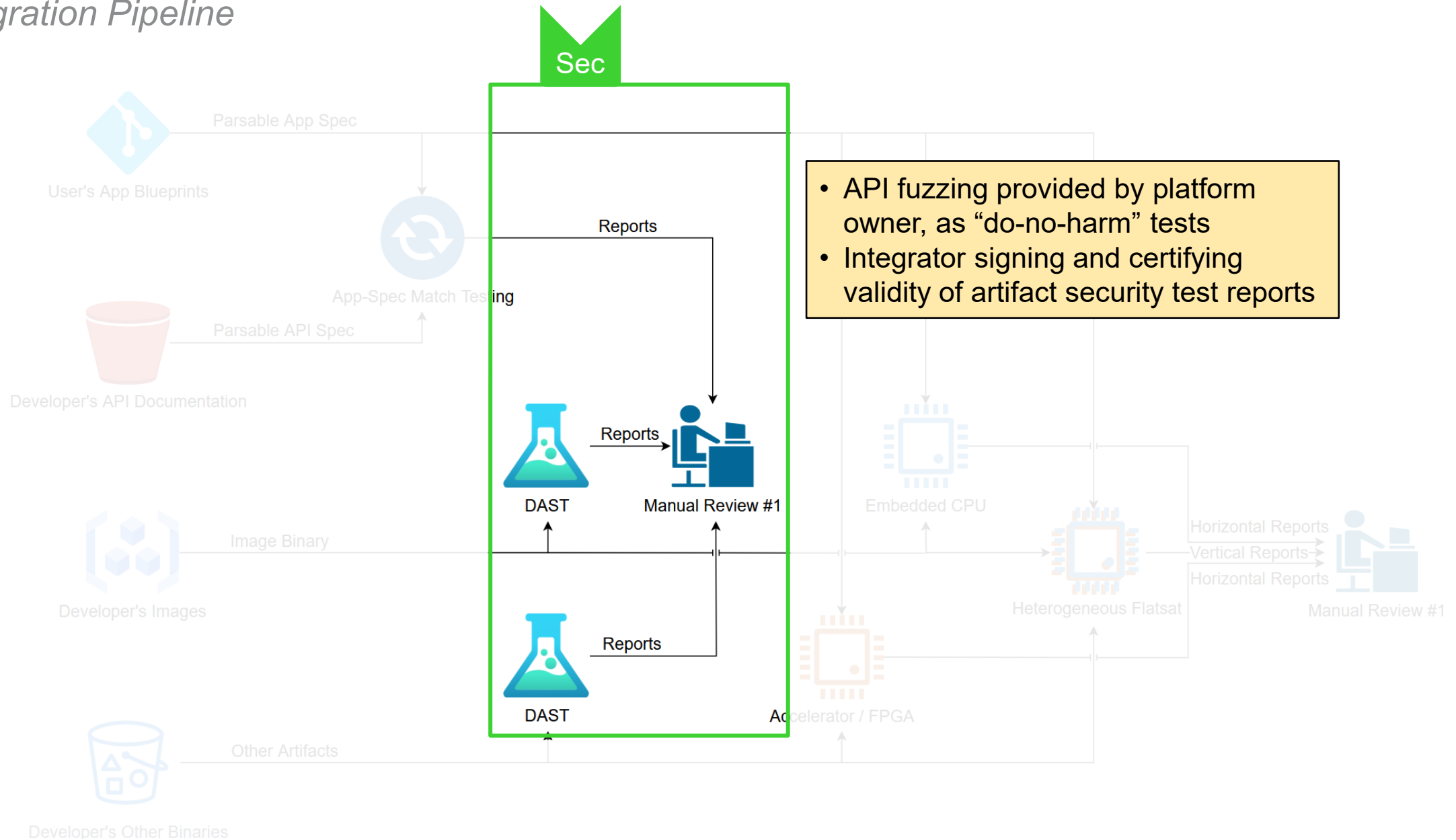
Dev





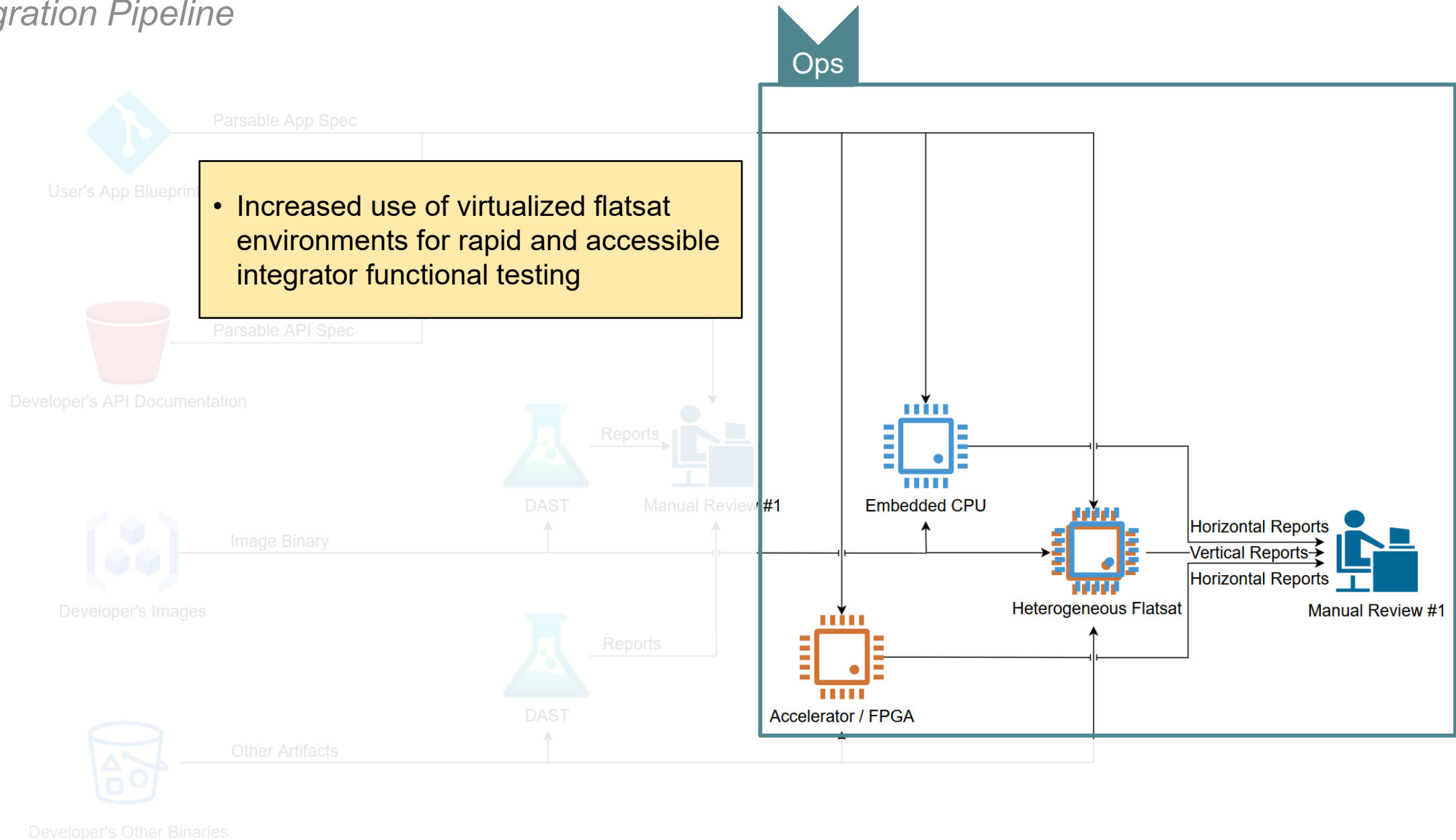
Reference Architecture

Integration Pipeline



Reference Architecture

Integration Pipeline





Conclusion

- Driven by increasingly dynamic user requests, modular reconfigurability will drive the adoption of app-based open systems architectures
- The Aerospace Corporation predicts many novel challenges for future ground systems in their support to highly reconfigurable “space cloud” satellites
 - Horizontal and vertical testing across virtualized complex mission compute payloads
 - FSW development cycles accelerated by multiple orders of magnitude
 - Sharp divide between developer and integrator roles
 - Memory security and isolation configurations on-asset to enforce zero-trust requirements



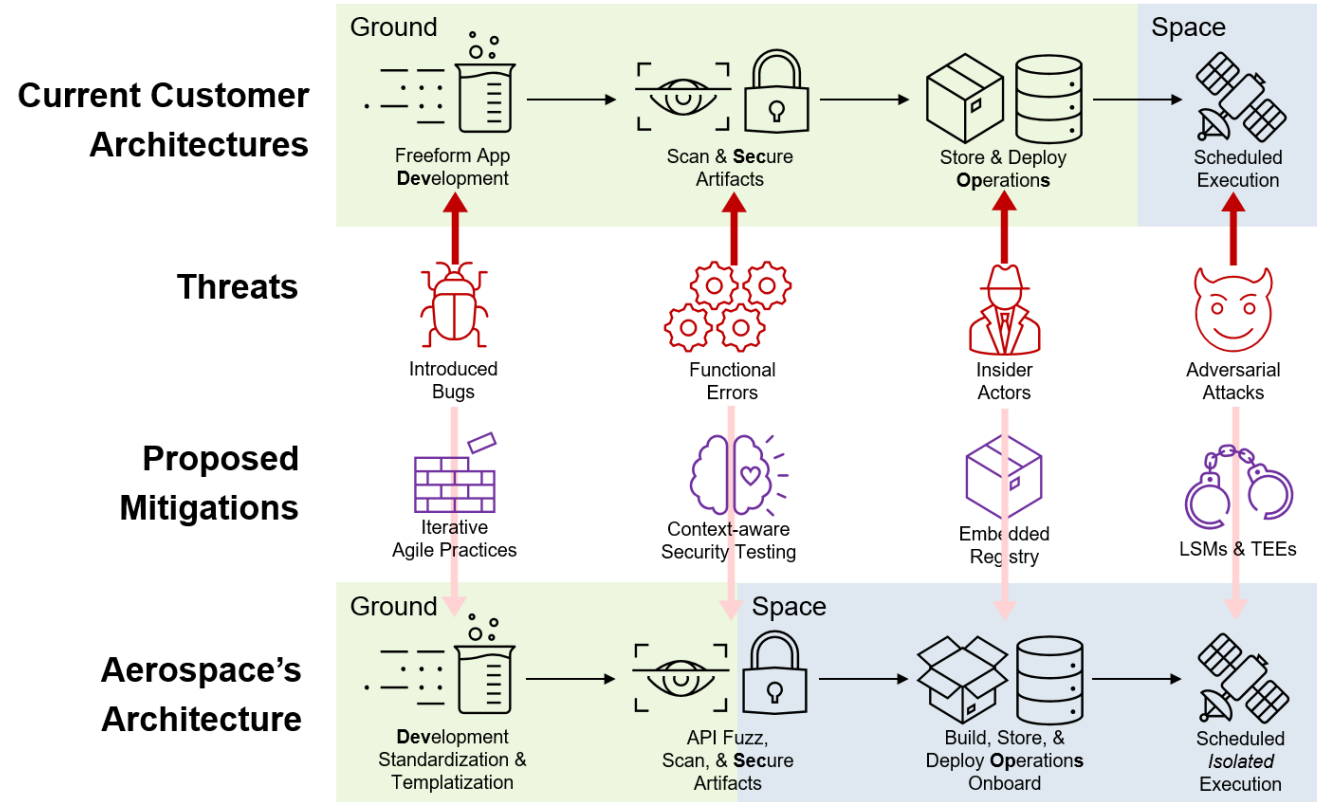
Backup



Onboard DevSecOps

Expanding the Pipeline to Space

- Container-based uploads can be too large for mission uplink.
- Adversarial compromise of ground system introduces new attack vector.
- Multiple payloads from various sources need to coexist securely.
 - Need for isolation between different classification levels (Top Secret, Secret, Unclassified, etc.)
- Aerospace has tested implementing the Security and Operational stages on space vehicles.
 - Pre-deployment security scanning
 - Building, storing, and deploying containers onboard space vehicle
 - Logical isolation tools during payload execution.



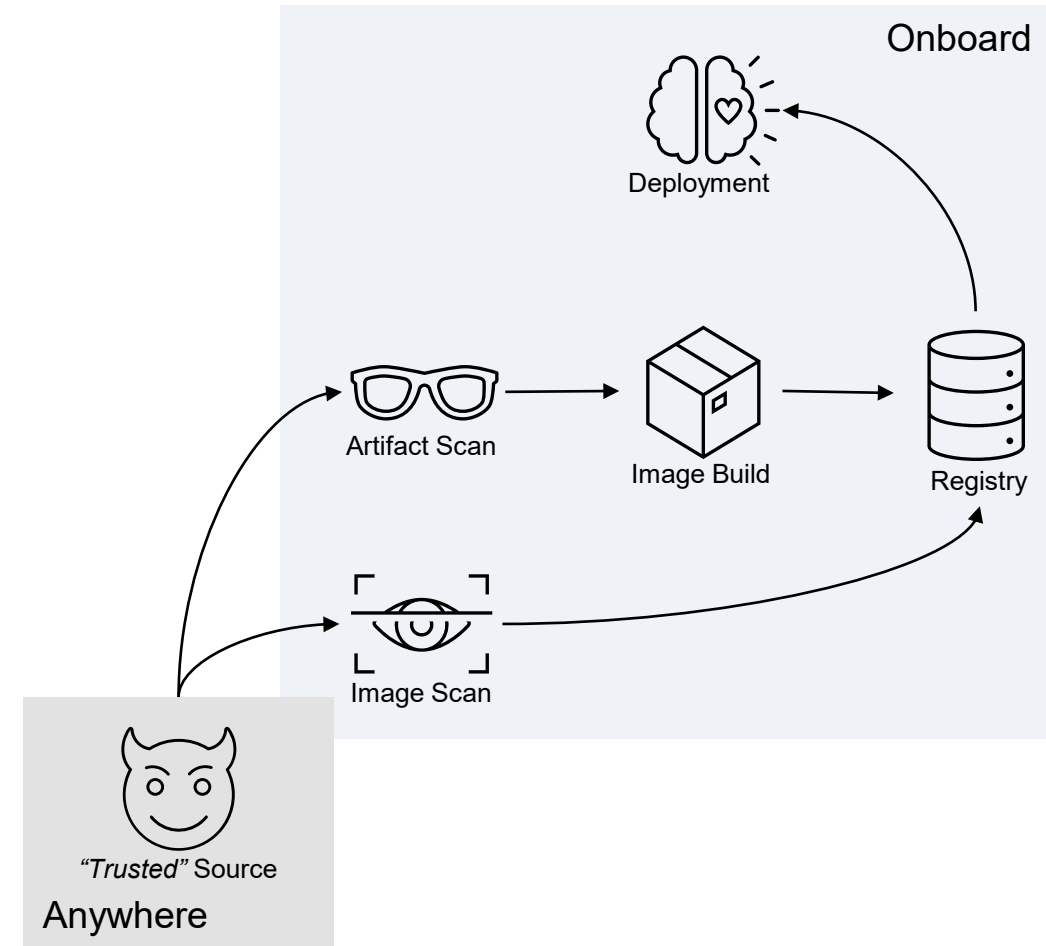
Current architectures perform all DevSecOps in ground before deployment. With the recognition of threats in all stages, our proposed architecture improves both ground and onboard DevSecOps speed and security



Onboard DevSecOps

Secure, Isolated Payload Execution in Space

- Containerized applications provide some isolation, but have existing flaws:
 - Enabling memory sharing removes some isolation.
 - Attacks on containers still exist.
- Experimental study on security scanning tools
 - Source-code scan tools for C, C++, and Python pre-compilation (Artifact Scan)
 - Container image scan tools for post-compilation (Image Scan)
- Trade study on common security isolation tools
 - Linux Security Module (LSM); fine-grained access control
 - Trusted Execution Environment (TEE); strong hardware-enforced logical isolation through offering like ARM TrustZone



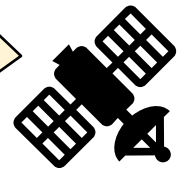
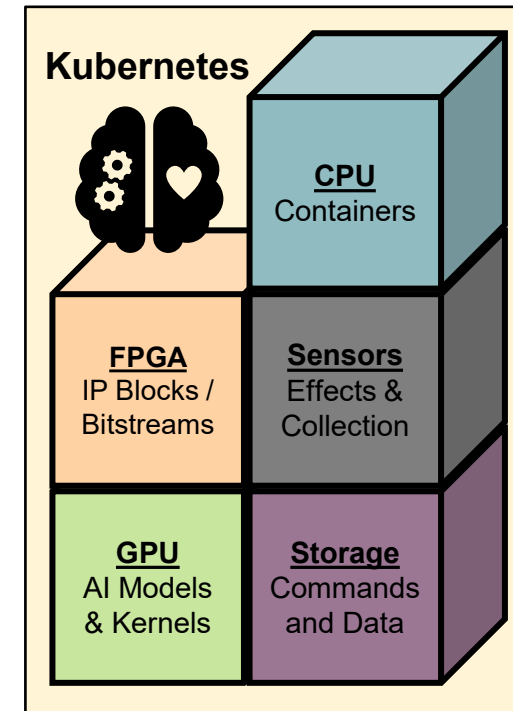
Vulnerability scan tools paired with strong logical isolation tools provide secure payload development and concurrent execution on-board space vehicles.



Aerospace's Dynamic & Distributed Space Cloud Architecture

Then vs Now

- Satellites today
 - Are programmed by a **selected** contractor
 - Can only rely on **themselves**
 - Require **squads of ground operators** for safe flight
 - Can only be used by a **few people**
 - Need to be told **exactly** what to do all the time
 - Largely **single-purpose** after launch
- Future satellites need to
 - Be programmable by **anyone**
 - **Cooperate** with other satellites
 - Operate with **less reliance** on humans
 - Receive the tasks **best** for them to execute
 - Autonomously determine **how** to fulfill requests
 - **Take over** for satellites suddenly out of commission

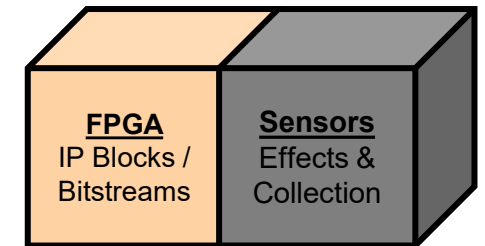
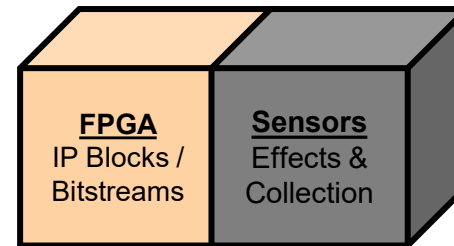
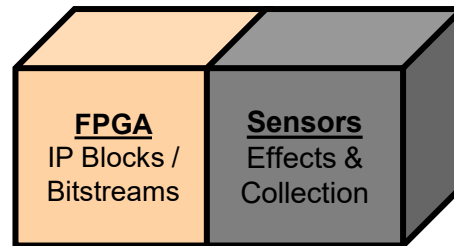
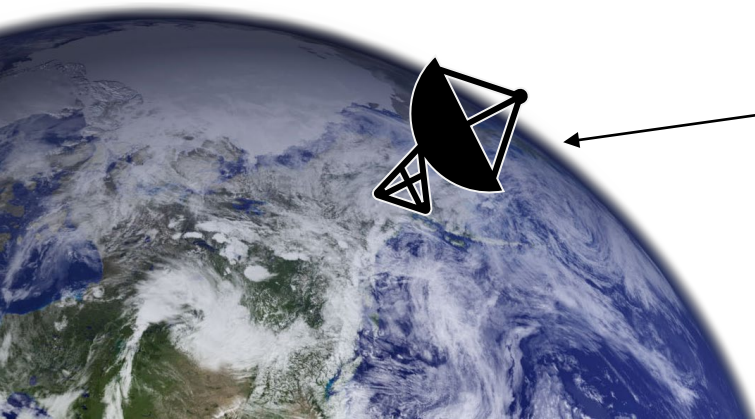


Pushing forward the Nation's compute in space by launching flying datacenters



Current Static & Stovepiped Satellite Architecture

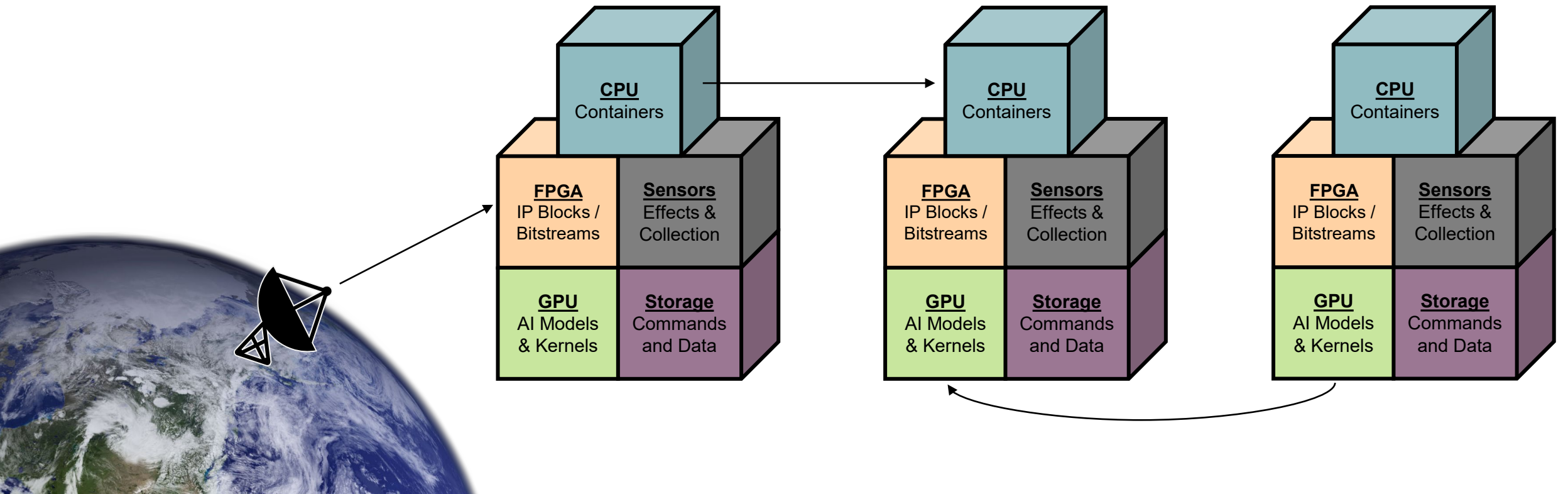
- Satellites today are
 - Largely **single-purpose** after launch
 - Programmed by a **single selected** contractor
 - Reliant on only **themselves**
 - **Explicitly commanded** by many humans
 - Comprised of **simple homogeneous** compute





Future Dynamic & Distributed Space Cloud Architecture

- Satellites today are
 - Largely **single-purpose** after launch
 - Programmed by a **single selected** contractor
 - Reliant on only **themselves**
 - **Explicitly commanded** by many humans
 - Comprised of **simple homogeneous** compute
- Future satellites need to
 - Execute on **new tasking** post-launch
 - Be programmable by **anyone**
 - **Cooperate** with other satellites, in part for resilience
 - Utilize **higher abstraction** & autonomous commanding
 - Utilize **complex heterogeneous** processors





Motivating Scenario

Then vs Now










- Reconfigurable Compute Scenario
 - A Satellite contains multiple compute types (Field-Programmable Gate Array (FPGA) blocks, GPU AI/ML Networks, CPU containers) and is running at 100%
 - New high priority task needs to be scheduled, requiring new compute type modules
 - New FPGA for specific mission critical fast processing using custom hardware design close to sensors
 - New GPU models to preprocess mission specific data translation to create more usable data reports
 - New CPU container requiring downlink software change to send data.
 - Requires an agile DevSecOps pipeline that can quickly develop safe modules for all compute types, handle packaging of payload, and verify operation in space-like environment.

Mission-critical reconfigurations must be possible rapidly and welcome module swapping between ground and space platforms



Nebula {X}

Research portfolio to de-risk rapid in-ops capability insertion and autonomy

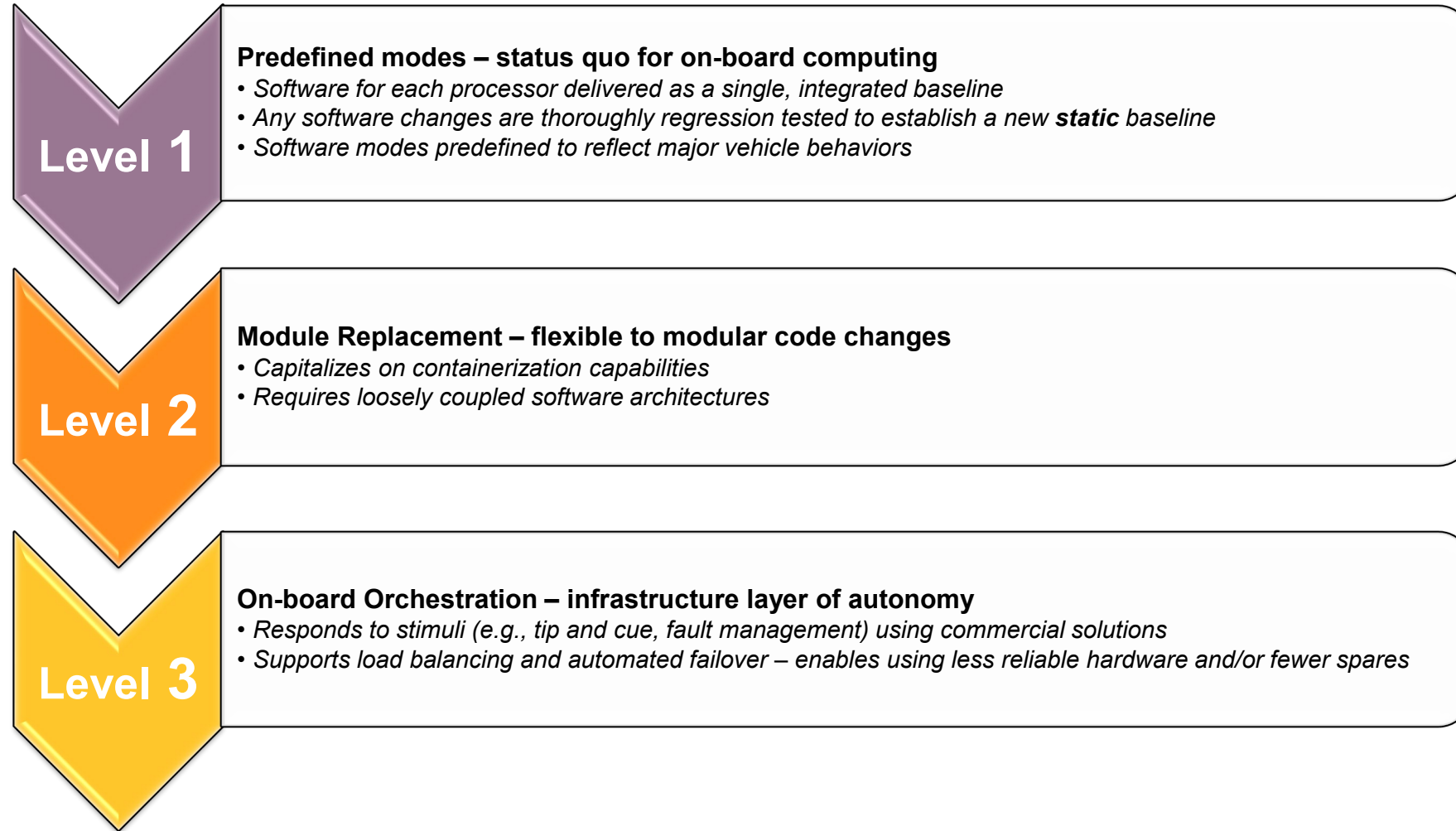
Design		NEBULA {XPU}
		NEBULA {C2}
		NEBULA {VEHICLE}
Engineering		NEBULA {DEVSECOPS}
		NEBULA {DE}
		NEBULA {CHAOS}
Mission		NEBULA {VISUALIZATION}
		NEBULA {GEOSPATIAL}
		NEBULA {PLANNING}

- Nebula {X}: collection of projects to address the technology gap multiple customers are rapidly approaching
 - Current focus on reconfiguration via containers and re-use of **commercial enterprise container orchestration software tools**
- Research goals
 - Maximize autonomy and resiliency of space processing
 - Blur the lines between “space” and “ground” capabilities
 - Minimize latency between tips and cues
 - Increase collaborative capabilities of intra- and inter-constellation efforts
 - Prepare customers for tracking and tasking increasingly larger counts of vehicles
 - Reduce reliance on human intervention
- Work ongoing since FY19; significant ramp-up in FY23
- **For more technical detail** on each project, see the [Nebula {X} Confluence](#)

Reconfiguration moves us from continuous production agility to continuous operation agility



Levels of Compute Reconfigurability



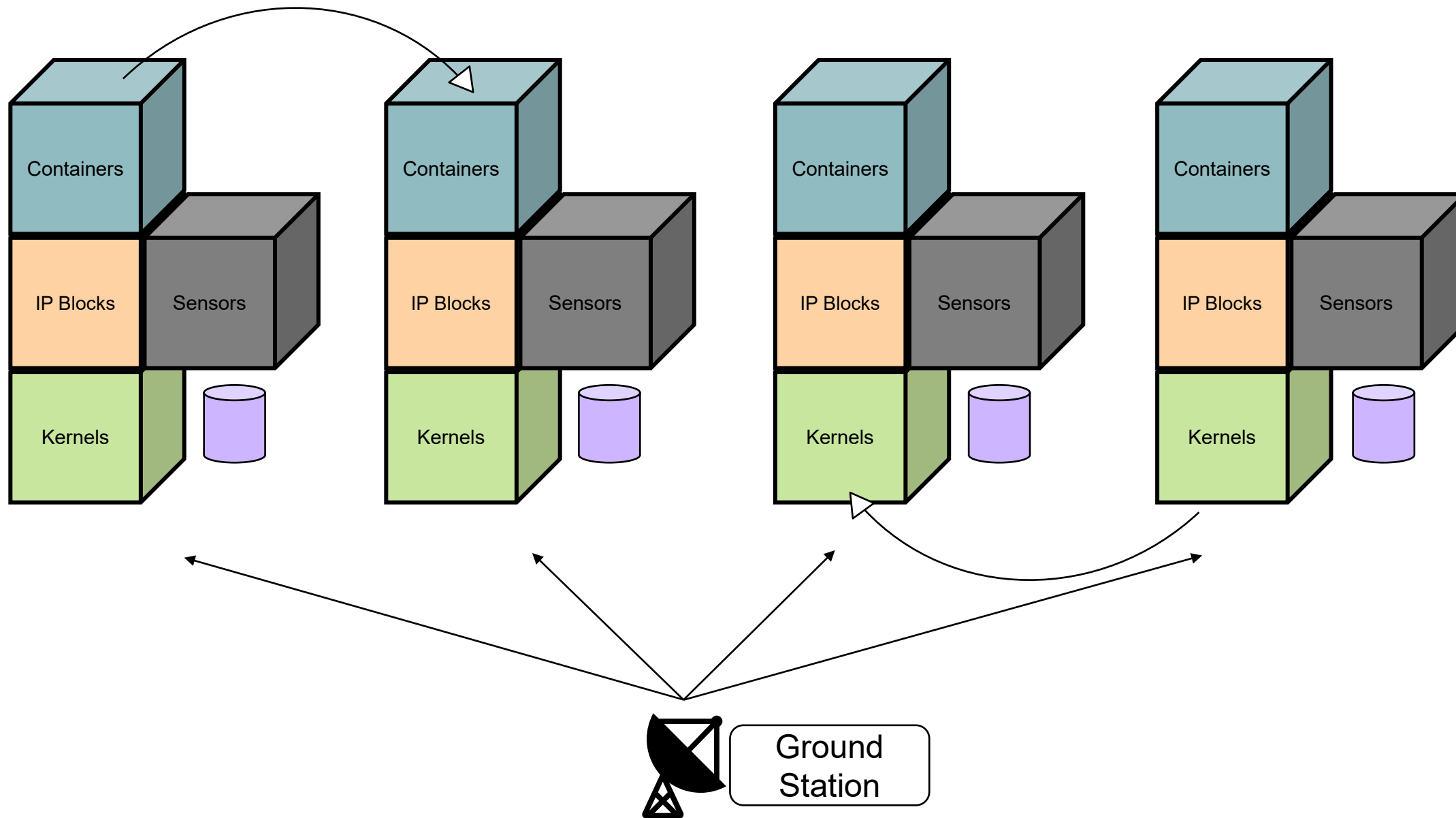
Reliable, but slow to change

Current Operational State of Autonomous Systems Research

Capability Gap for Future Need

Every advanced technological capability for future architectures will require Level 3 orchestration as a foundation

Reconfigurable Compute Diagram: Module Swapping

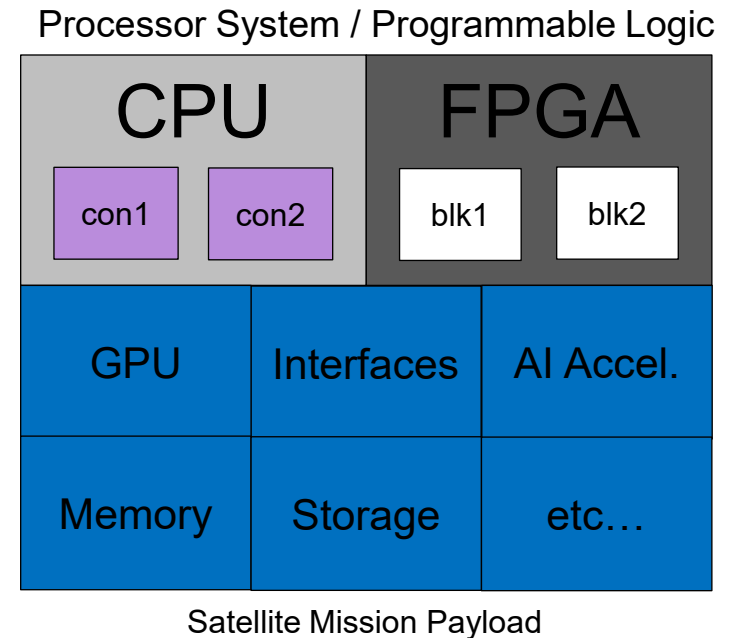




Background

Enabling Technology: Heterogeneous Compute

- CPU, FPGA, and other ASIC HW devices are often integrated as part of a single payload
 - A System on Chip (SoC) integrates these die into a single package
 - These COTS parts are highly SWaP-C2 optimized
- Containers themselves only run on CPUs, as only CPUs can support a general purpose kernel / host OS
 - The container running on a CPU may both configure and exchange data with other components on an SoC
- Independently reconfigurable modules are also possible within FPGA fabric (like containers but in FPGA fabric)
 - Xilinx Dynamic Function eXchange
 - Ettus RF Network on Chip



Future DevSecOps tooling will require pipeline standardization for compute elements apart from CPU containers



Reference Architecture

Development Pipeline

Dev

Develop app source code from SDK and compile into executable binary

1. Template-based design to support short-iteration “Agile” development
2. Emphasis on reuse of existing software architecture and compiled artifacts
3. Model-Based Software Engineering (MBSE) techniques and Linters

Sec

Secure source code and generate outgoing reports

1. Static Application Security Testing (SAST): OWASP & MITRE CVE/CWE scanning
2. Wide code coverage of different types
3. Manual architecture security review for fault isolation and error propagation

Ops

Test on mock pseudo-Operational environment using given API calls

1. Base HW interface testing
2. Cloud sandbox deployment
3. Performance testing in clean environments

Low-classification application development and testing – isolated



CPU Containers

- Host and develop mission app source code
 - Quickly swap out mission-critical application to meet new high-priority task and run on general-purpose CPU
 - Quality testing code with unit tests that meet code coverage.
- Pre and Post Compilation security and vulnerability analysis
 - Ensure new application is safe for satellite deployment.
 - Source code and object scanning to detect vulnerabilities.
- Functional testing before packaging into deployable image ready for integration.
 - Sandbox testing of mission environment to ensure application performs task for specific end-user.
 - Ensure container internal sources can interface with container.



FPGA Bitstream

- Develop design specific FPGA bitstreams for changing hardware requirements in different mission specific tasks.
- Promote agile development for ground systems
 - Provide developer pipeline with templated FPGA designs that have been tested and verified for integrator pipeline and operational environment
 - Speed up development process with templates.
- Hardware Design Language (HDL) security scanning pre-compilation to ensure bitstreams do not have known vulnerabilities
- Post-compilation verification to ensure bitstream generation does not contain vulnerabilities or was maliciously altered.
- Simulation testing of hardware design to validate performance and ensure functional operation.

GPU AI/ML Networks



- AI Networks that take trained models and mission specific data with untrained models to aid in preprocessing data
 - Can speed up and improve mission specific data translation by preprocessing data through AI Model using GPU.
- GPU is often neglected as compute type for space payload
 - Developer pipeline can take pre-trained models to optimize operations once deployed



Reference Architecture

Integration Pipeline

Dev

Develop app wrapper and infrastructure SW, if necessary

1. SDK and API definition
2. Infrastructure application development
3. Penetration testing and red-teaming

Sec

Secure interfaces between apps and hardware, and verify reports

1. Static, Dynamic & Feedback-based Application Security Testing (SAST/DAST/FAST) and API Fuzzing
2. Least privilege execution during runtime
3. Memory, Storage, and Inter-Process-Communication (IPC) interface verification

Ops

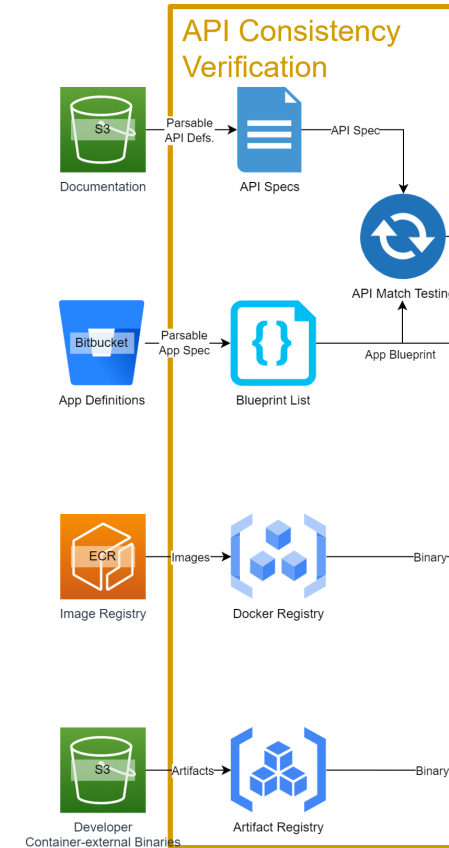
End-to-end tests on representative Operational environment and commit

1. Input mission data generator to system
2. Output analysis data verifier from system
3. Low-latency delivery system to mission management center

High-classification infrastructure integration efforts – context aware

Integrator Pipeline Development Stage

- API consistency verification to ensure incoming artifacts adhere to specifications and meets end-product specifications.
- Ensure that payloads coming from various sources to a single target match API requirements and contain development features for specific
 - Mission application in Docker images verified against docker registry, ensuring base layers are up-to-date and meet specifications
 - FPGA bitstream artifacts checked against integrator-hosted registry.
 - GPU/AI code and models verified to ensure input/output formats are compatible with infrastructure and GPU-specific functionalities behave consistently

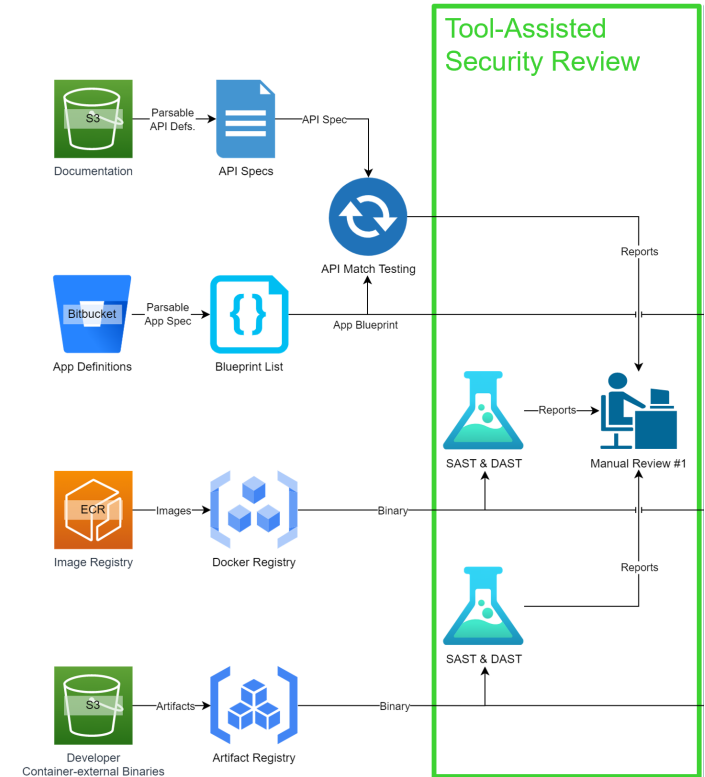


Develop app wrapper and infrastructure SW, if necessary



Integrator Pipeline Security Stage

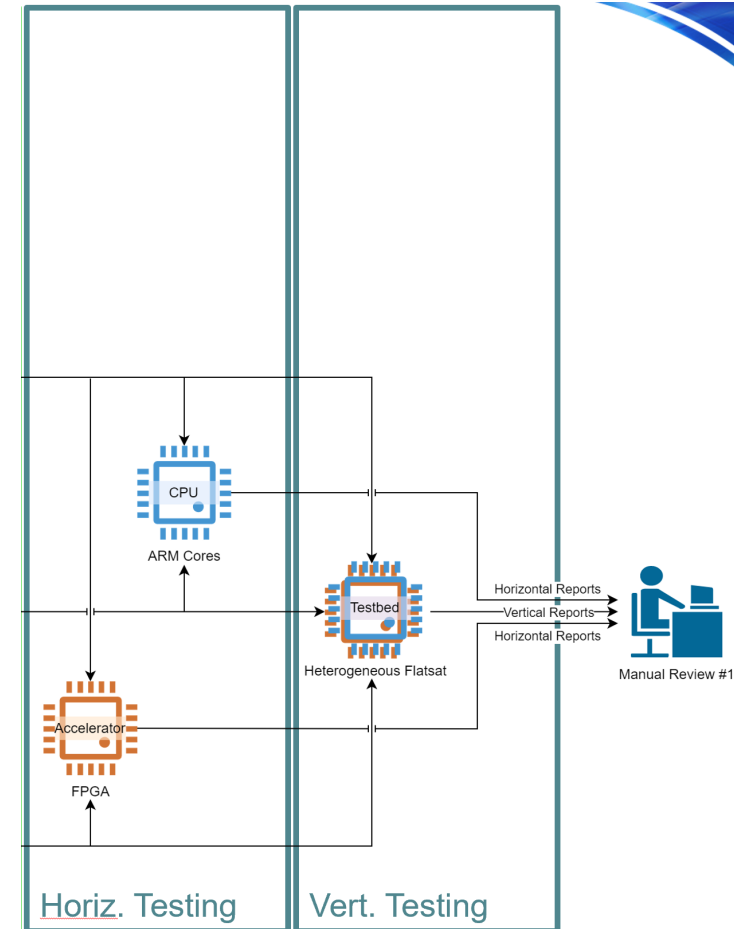
- Ensure incoming compute type artifacts adhere to proper security and have been adequately tested for vulnerabilities and potential errors
 - API fuzz testing with invalid, malicious, or error-inducing inputs to determine potential vulnerabilities.
 - Fuzz testing to determine how API handles security protocols.
- SAST/DAST used on container, FPGA, and GPU accelerator artifacts from developer pipeline.
 - Dynamic environments to run and check for vulnerabilities of mission-critical applications in Docker containers
 - Hardware emulators of mission systems to test incoming FPGA bitstreams
 - GPU application that processes sensitive data use DAST tools to determine how application performs under different workloads and detecting runtime vulnerabilities
 - FAST absorbs results from SAST/DAST on all tools to improve security for future artifacts
- While relying less on users, security stage still uses manual review for result verification



Secure interfaces between apps and hardware, and verify reports

Integrator Pipeline Operational Stage

- Preparing the payload for integration to the operational environment with integrator pipeline testing
- Horizontal testing
 - Testing each artifact individually for functional accuracy and seamless integration of new artifacts.
 - CPU Containers tested against each other <add more info>
 - FPGA blocks tested against each other, verifying input-output behavior, data paths, and interfaces to achieve expected outputs.
 - GPU <add more info>
- Vertical testing
 - Operational testbed environment used to test compute type functionality with each other.
 - Mission-specific CPU application, hardware FPGA design, and GPU AI/ML models ran on satellite-representative testbed
 - Ensure all compute types properly operate before deploying as payload.



End-to-end tests on representative Operational environment and commit



Security Scan Tools

- Determine which security tools would be appropriate to run on a spacecraft.
 - Build data scan tools used to scan C, C++, and Python source code being built into container images
 - Image scan tools used to scan vulnerabilities on built images
- Many security tools are available for developers, however some of them are computationally expensive, have high overhead, or require to be constantly connected online for updates or database requests.
 - Finding tools that satisfied our requirements: running on an ARM64 embedded device, avoiding frequent updates, performing well offline.

Build Data Scan Tools

- PyCQA Bandit – Python Scanner
- Wheeler Flawfinder – C/C++ Scanner
- ZupIT Horusec – Multi-language tool

- Single-language tools were significantly faster at finding vulnerabilities
- Single-language tools had same vulnerability coverage

Image Scan Tools

- Gype
- Trivy

- Trivy and Gype detected a similar number of vulnerabilities in almost all cases
- Gype ran faster on average, but Trivy had lower standard deviation
- Gype used reduced disk size compared to Trivy

Onboard security scanning trade study using spacecraft-friendly tools



Thank you