CLOUD-BASED PRODUCT GENERATION PLATFORM – LESSONS LEARNED

JAMES GUNDY AND JUSTIN SANCHEZ

Senior Software Engineers

HARRIS.COM | #HARRISCORP

HARRIS® TECHNOLOGY TO CONNECT, INFORM AND PROTECT™

Goals & Objectives

Harris has been a long-standing mission partner with NOAA in developing and deploying mission critical Earth-sensing instruments and ground systems

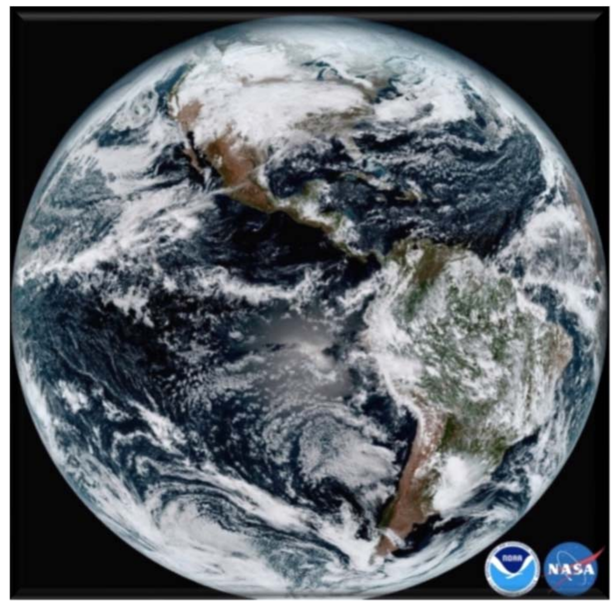
- Harris has leveraged the successful Product Generation architecture from GOES-R to develop the next-generation distributed product processing infrastructure called Downburst™
- This briefing presents lessons learned from Harris research and development for Cloud-based product processing

Distributed Product Processing Infrastructure:

- Dynamic, parallel block processing for scalable, high-performance computing
- In-memory database for high-throughput input/output (I/O)
- High-speed messaging system
- Multi-mission support

Derived from GOES-R Product Generation Architecture

- Leverage technology and approaches used for product processing workload of five satellite instruments
  - ABI with 16 bands and 2km - 0.5km resolution
  - Generates 35 L0/L1/L2+ environmental and space weather products from geostationary satellite
  - Multi-regional processing – full disk, CONUS and mesoscale (non-fixed location)
  - 100Mbps raw data rate
  - Generates 16.1 TB products per day (60x more data than previous generation)



GOES-R First Light Image (True Color)

Downburst™ Features and Characteristics

Compute Intensive	• Parallelization to satisfy tight product latencies • Distributed processing across 200+ servers
High Throughput	• 697,168 files and 16.1TB data per day • Latencies as low as 1.8s
High Reliability	• System availability 99.99% • Product availability of 99.9%
Adaptability	• Complex product dependency model • Capability to add/update algorithms at run-time
Scalability	• Scale 300+% without redesign
Security	• FISMA high

Transition Downburst™ to Cloud

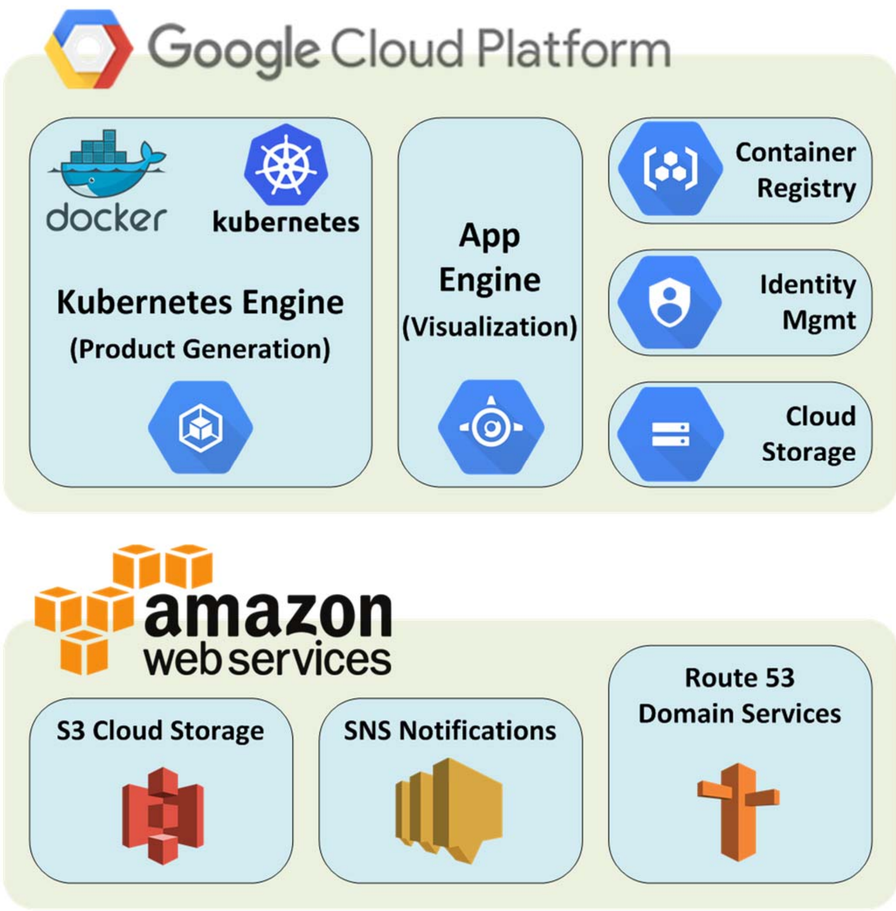
Research Goal: Demonstrate Downburst is cloud compatible

Drivers for moving to the Cloud:

- Reduce infrastructure costs
- Ease scalability
- Improve maintainability
- Relieve facilities constraints

Public Cloud

- Current utilization of multiple cloud vendors (Google and Amazon)
- Fully containerized solution using Docker and Kubernetes
- Distributed architecture providing straightforward transition to cloud
- Location in multiple regions
  - Asia-east for Asian satellite data
  - US-east for US satellite data



Cloud Paradigm Changes

More focus on mission, less on infrastructure

- Engineering talent focused on developing/running services
- Infrastructure/hardware administration effort significantly reduced
  - Manpower maintaining local infrastructure would exceed cloud cost alone

Fluid Compute Resources

- Get resources that you need, when you need it
  - Expand the resources for extra missions/testing on demand
  - Run in the region that is best fits mission need
- Reduce cloud costs by deleting resources on off-hours
  - Forced team to script/automate all parts of deployment/teardown
  - Created consistency and quality of deployment/teardown (10-15 Minutes)

Increased accessibility

- Engineering talent not restricted to working a specific location
- Accessing resources and standing up demonstrations is easier
- No impact from local shutdowns enables greater up-time

Lessons Learned – General

Transition was fairly straightforward – no significant roadblocks

- Initial port only took a few weeks (proof of concept)
- Downburst™ similarity to microservices architecture facilitated smooth transition to Docker/Kubernetes
- Use of Googles Kubernetes Service (GKS) minimized infrastructure management

Google Cloud Platform (GCP) was bleeding edge in the beginning

- Significant changes in interfaces and commands encountered over the year
- GitHub projects/tutorials that leveraged GCP become outdated over time

Constant security awareness was needed

- Virtual machines are deployed securely by default, but could easily be made unsecure by opening firewall ports, exposing service IPs
- All traffic was routed through Kubernetes Ingress Controller to restrict number of open connections
- Secured connects facilitated through Let's Encrypt + Oauth2 authentication

Lessons Learned – General (continued)

Storage management was complex

- Used storage buckets for products
  - Access was either project-restricted or public, increasing difficulty in controlling access
  - Products were regularly purged to control cost
  - Required administration to manage purges effectively
- Often still required virtual disks for applications
  - If configuration not properly set, new disks automatically were created, but not deleted automatically
- Used Gluster for shared disk storage
  - Built in Kubernetes storage could not be shared across multiple services
  - Gluster/Ceph must be setup manually – not difficult to setup, but challenging to automate

Docker images were controlled in our own repository

- Major upgrades in public images can cause issues unexpectedly
- Improved control of contents inside images

Lessons Learned - Kubernetes

Kubernetes provides container orchestration

- Resource Management
- Horizontal Scaling
- Controlled Rollouts/Rollbacks
- Networking/Load Balancing
- Configuration Management
- Storage Access/Management
- Cloud Portability
- Open Source

Kubernetes has a steep initial learning curve, but can provide significant value if utilized fully

• Deployment and StatefulSet for deploying images/pods	• Is more resilient and scalable than simple pods
• PersistentVolume/Claim for storage configuration	• Abstracts persistence deployment • Improves management of storage resources
• IngressControllers in service configurations	• Performs all routing in Ingress Configuration - simpler than custom proxies
• ConfigMaps and Secrets for configuration management	• Easier to manage than persistent volumes • Secrets obfuscate sensitive information - not really secure without RBAC
• Readiness and Liveness Probes for monitoring	• Determines when pods have completed startup • Necessary to account for dependencies in automated deployment