Utilizing DevOps in Cargotec IoT Cloud

Juha Uola
DevOps Lead

Digital Solutions Hub’s main goal is to provide software products and services support to all Cargotec business units.

DiSH also owns and develops Cargotec IoT Cloud.

CARGOTEC DiSH

KALMAR MACGREGOR HIAB

Cargotec Business Units

MACGREGOR

KALMAR

HIAB

navis

BROMMA

CARGOTEC DiSH

Cargotec IoT Cloud

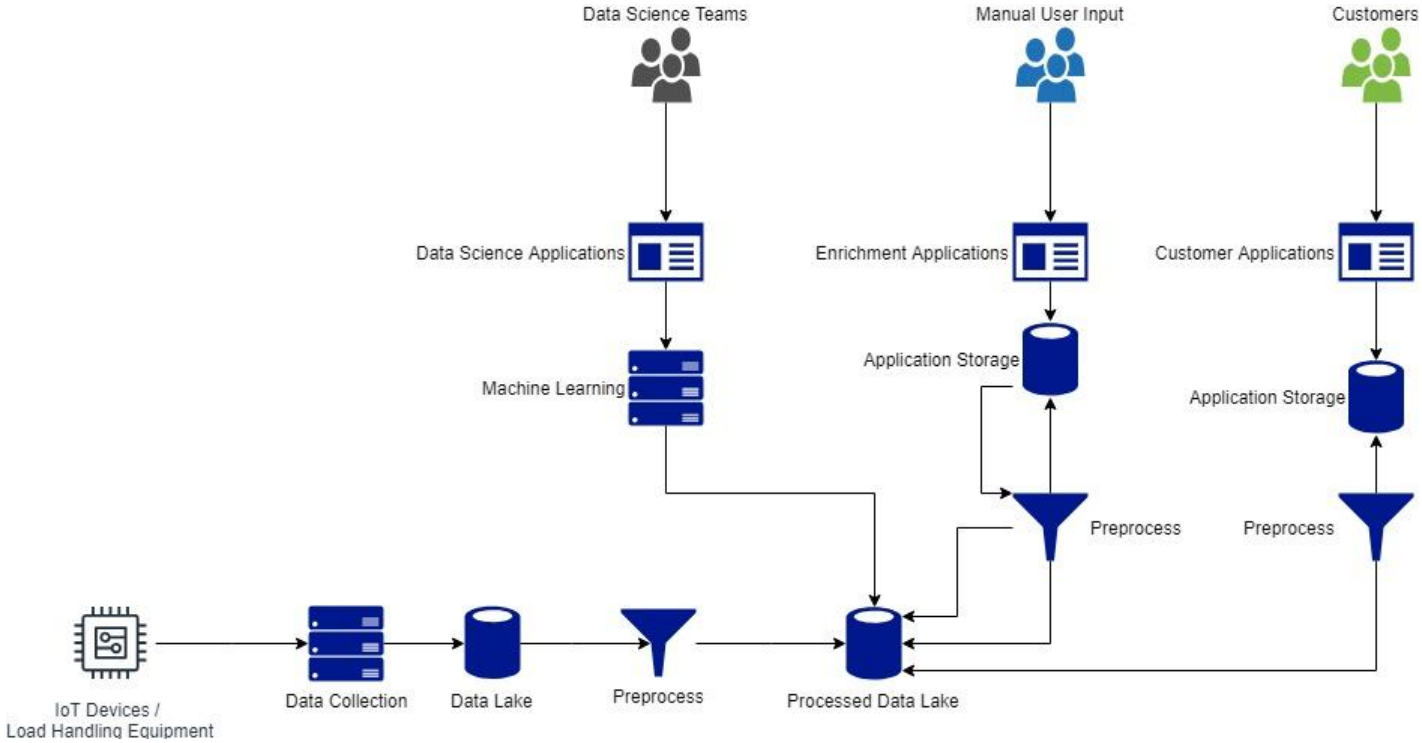
An AWS based cloud platform for ingesting, processing, storing and retrieving data from various sources. Making this data available for building new customer applications.

Data figures

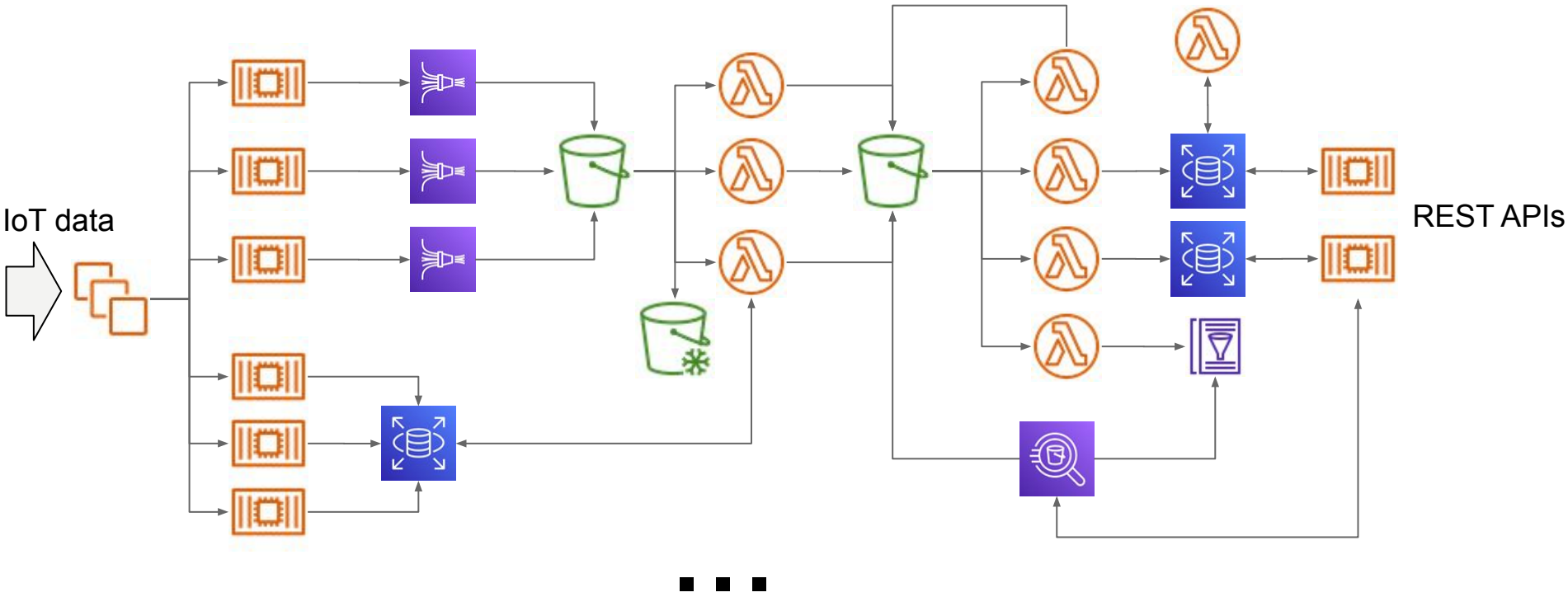
- Billions of rows / month
- ~200GiB / month
- ~9000 devices sending data

CARGOTEC DiSH

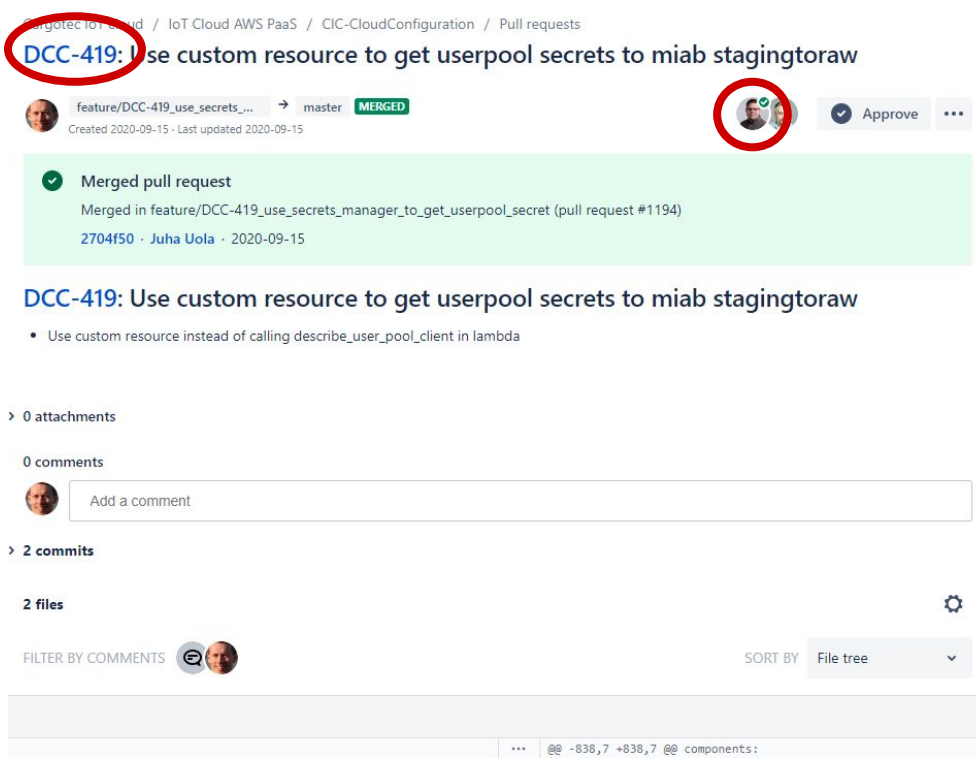
Cargotec IoT Cloud

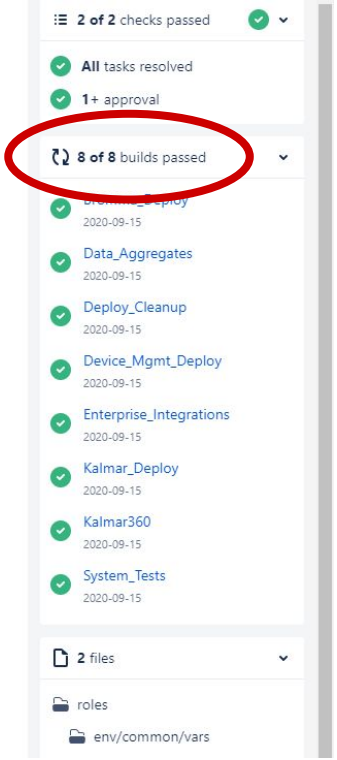


Architecture example



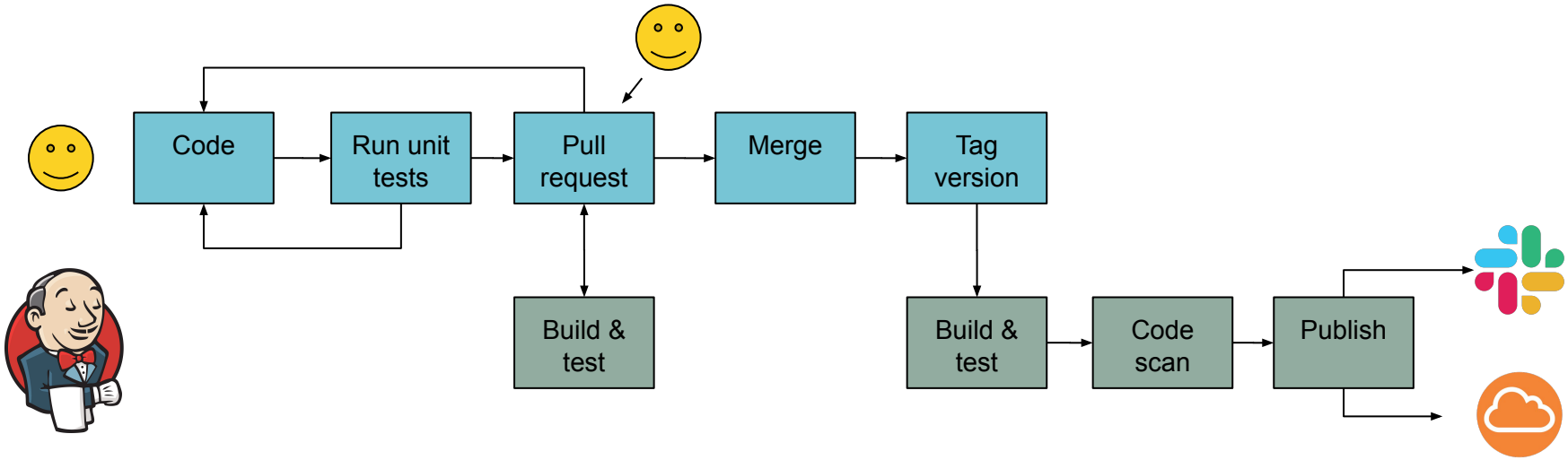
DevOps developer support





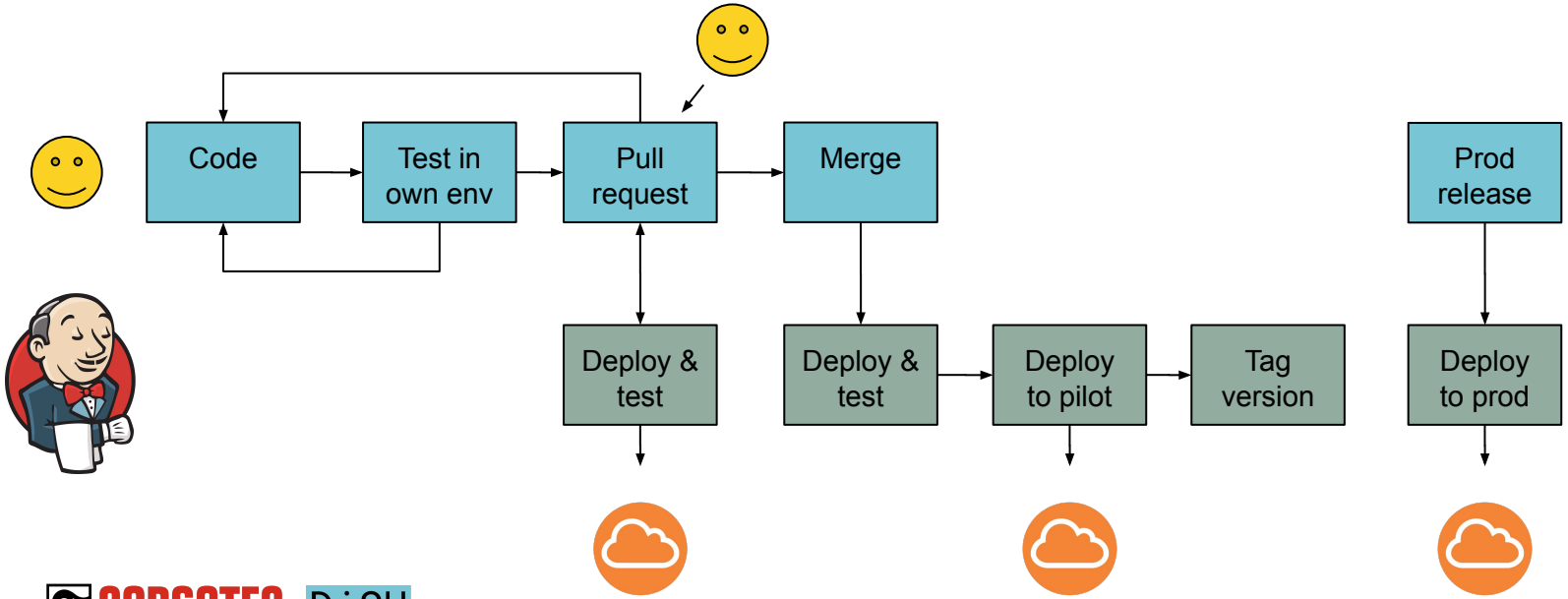
DevOps building block 1: Automated testing - unit tests

- All components should have unit tests
- Unit tests are ran by CI pipeline



DevOps building block 1: Automated testing - system tests

- Starting point: Basic system smoke test
- Goal: All main features covered with system tests
- Running system tests needs to be part of the CI pipeline
- Bonus: Developers are able to run system tests against their private test environment



CARGOTEC DiSH

Learnings from automated testing

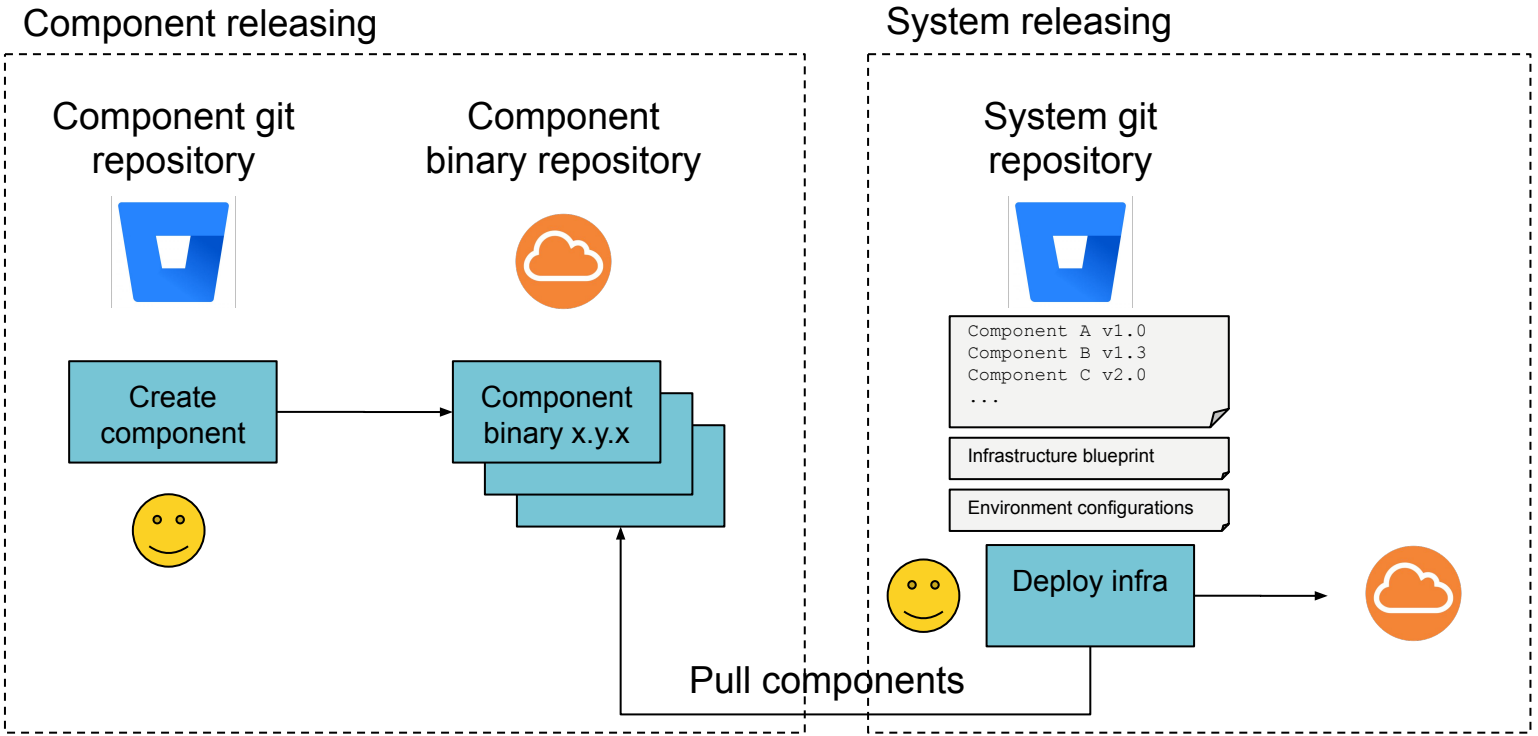
- Automated testing helps catching bugs early
- Tests need to be reliable for automation to work
  - Introducing new tests is a challenge for reliability
  - System test problems affect all developers
- Tests need to be ran in controlled environment
  - Jenkins build agents only have Docker installed
  - For system tests the whole test cloud environment should be created from scratch for each test run, but this is rarely possible in practise
  - Need to be able to at least update test cloud environment with desired changes
- Test results for pull request should come under 15 mins
  - May need to test in parallel and reduce pull request test set
- There are many build pipelines, those should be created as code to allow code reuse
- Tools that we use
  - BitBucket for git repository
  - Jenkins for build automation
  - Whitesource for security scanning
  - Robot Framework for system tests

DevOps building block 2: Repeatable cloud deployment

- Automated system testing requires that cloud environments can be created, updated and removed easily
- Repeatable cloud deployment requires
  - All software components are versioned
  - Cloud infrastructure is defined as code
  - Configurations for different environments, such as testing, pilot and production, are defined as code
  - DB schema changes are defined as code

```
docker-compose run ansible_playbook create_infra.yml -e "project=iotcloud env=testing stackName=myStack"
```

Repeatable cloud deployment

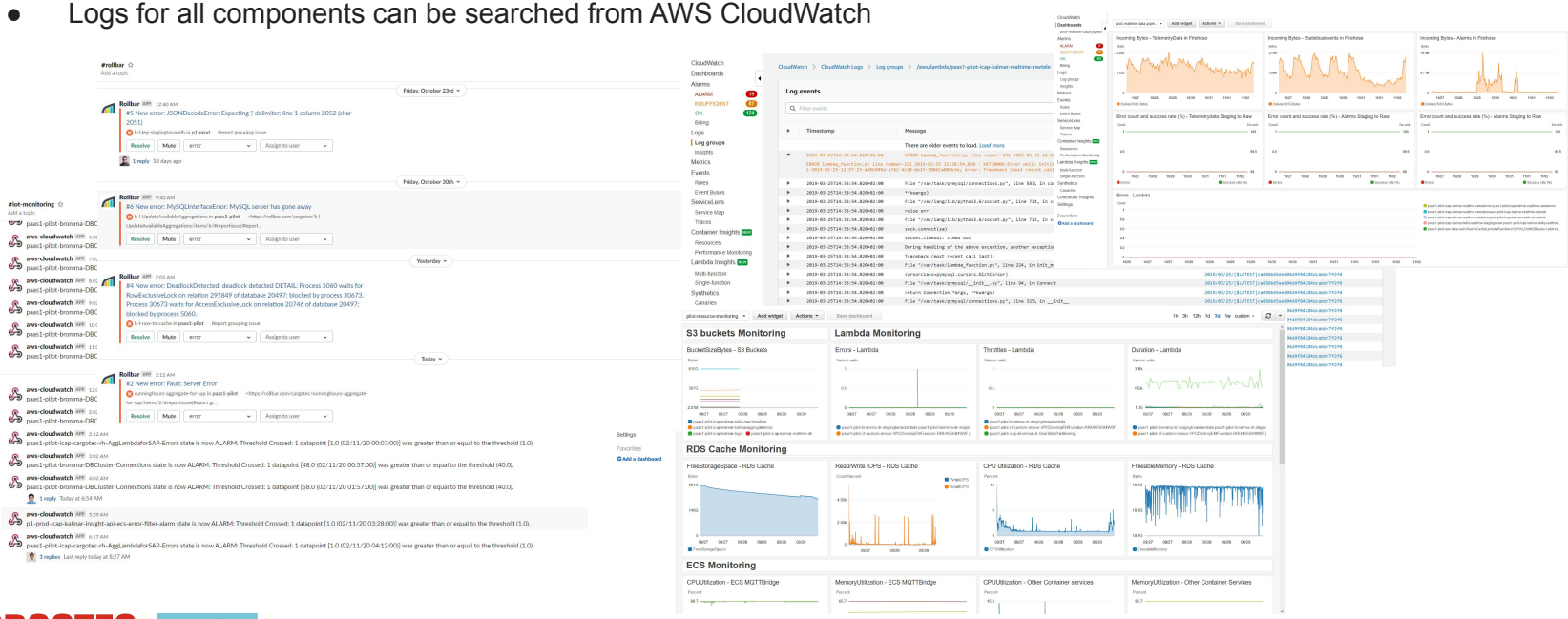


Learnings from repeatable cloud deployment

- Cloud deployment from scratch must pass system tests
- Cloud deployment from scratch takes several hours, updating is faster
- Manual operations are an exception in pilot and production, typically only CI pipeline is updating them
- Being able to create own test cloud is a treat for developers
  - Can be expensive, so need to downsize all possible resources and optimize availability
  - Cloud environment removal is important for developers, but it can break easily
    - Periodical CI job to test cloud creation, system tests and cloud removal
- Continuous delivery has been discussed
  - Component release could result a cloud deployment
  - Would require further emphasis on system testing
  - No requirements for high update frequency
- Technologies that we use
  - AWS CloudFormation + Jinja2 for infra definitions
  - Ansible for managing configurations and variables

DevOps building block 3: System monitoring

- Rotating support team within the development team
- Support team is alerted in Slack, if system is unhealthy
- AWS CloudWatch dashboards are used to debug the discovered issues
- Logs for all components can be searched from AWS CloudWatch



CARGOTEC DiSH

Learnings from system monitoring

- System monitoring is required, if issues are to be found before customer starts complaining
- System health alerts are sent to dedicated Slack channel, but this can get noisy and hard to follow
  - Dedicated error monitoring tools such as Rollbar can help to categorize issues
- Technologies that we use
  - AWS CloudWatch for logs and dashboards (planning to use 3rd party tool)
  - Rollbar for keeping track of application errors

IoT Cloud DevOps journey

From

- Manually set up servers and infrastructure
- EC2-based architecture
- Manually maintained development, pilot and production clouds with different configurations
- Changing and deploying components directly in production
- Manually testing the system after each change (or not at all)
- Each developer having their own components that nobody else knows about
- Each component deployed differently
- No component versioning
- No system versioning
- No automated system testing
- Errors are detected by customer

=> Development team spending most of its time figuring out why system is not working and what broke it

To

- Infrastructure as code
- Cloud-native architecture
- Setups can be checked from source code
- Similar development, pilot and production clouds
- Managed cloud environments are updated only by changes in source code
- All code is peer-reviewed, which spreads the knowledge
- Automated system tests
- Components are versioned and published is similar fashion
- System is versioned and test results for each version are available
- System monitoring alerts to Slack
- Errors are detected by the development team

=> Development team spending most of its time developing new features and creating new tests

Iot Cloud DevOps next goals

- Monitor also data quality, not just data ingestion operation
- Shorten development cycle time
- Improve system deployment time
- Improve system tests coverage, reliability and performance
- System performance testing
- Make development process status more visible for stakeholders

IoT Cloud recent failures

- Updating a library version inside component A caused change in component A output, other system components were unable to read the new format -> 100 days of aggregated IoT data corrupted
  - Cause: Missing system test covering component A
  - Solution: Revert the change, recalculate data, create system test
- Change in cloud infrastructure code caused pilot deployment to fail even though system tests in dev environment passed
  - Cause: Pilot environment had old manual configuration that prevented the update
  - Solution: Remove manual configuration and add corresponding infrastructure code to master branch

Thank you! 😊

For more information about DiSH,
Jobs and trainee positions

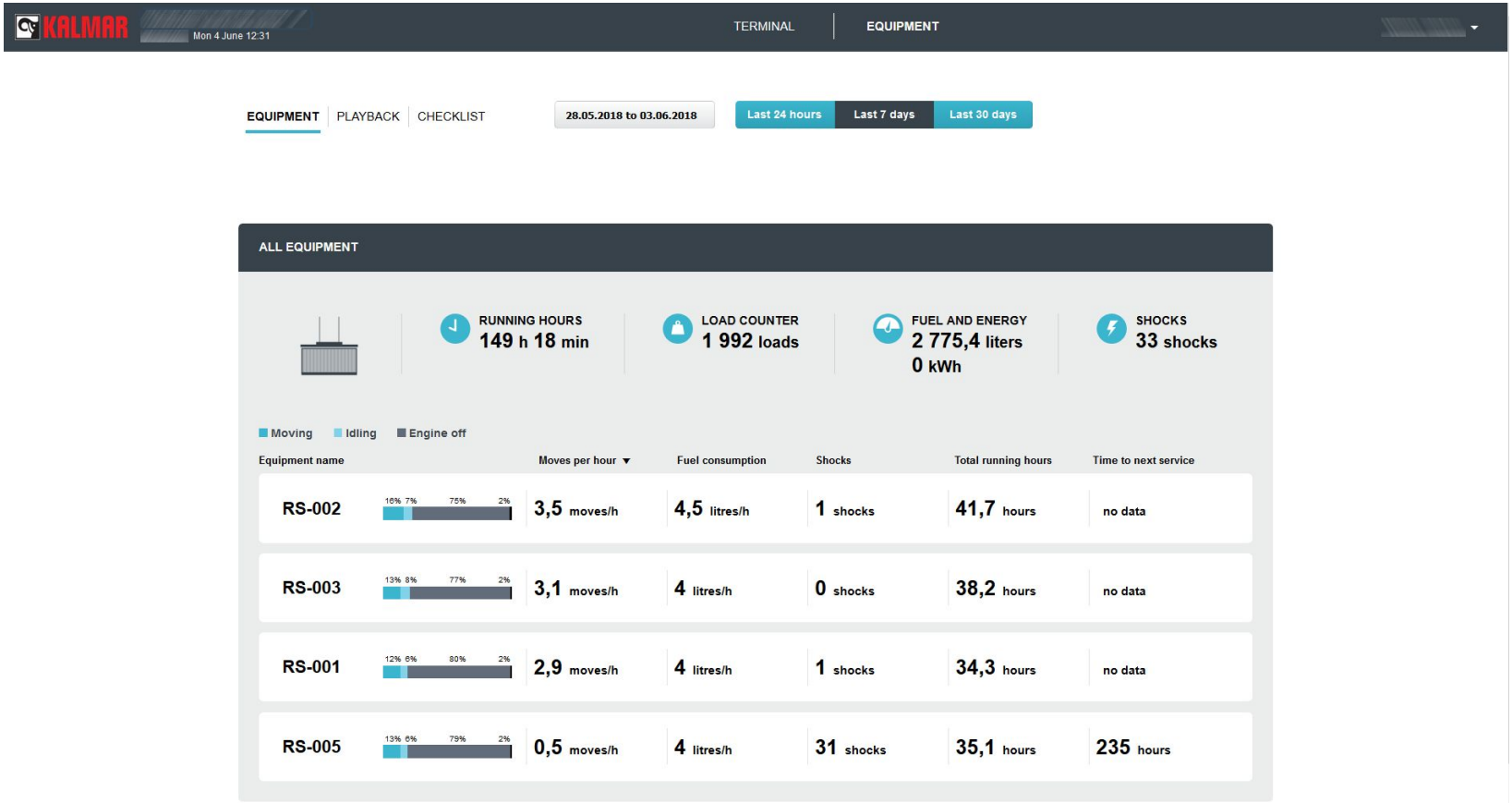
Juha Uola
juha.uola@cargotec.com

Petri Selonen
petri.selonen@cargotec.com

CARGOTEC DiSH

Appendix

CARGOTEC DiSH



CARGOTEC DiSH