I will do the presentation for SHIP 2.0 migration as SWE is on leave today.

Here is the agenda,

We will recap for previous release and introduce the migration to gitlab which we will share the current migration status and steps we did for all our repo.

And we will share the stages we build for our continuous integration pipeline workflow.

In the end, we will share our plan for later release.

In release 1.5

We have evaluated DevSecOps tools recommended by SHIP 2.0 ​

* Test CI pipeline in Gitlab for automation​
* Create POC for migration of SRE repository and pipeline​
* Prepare migration plan and draw the roadmap​.

And in release 1.6

We Onboarded to SHIP-HATS and TechBiz subscription

And base on the research we did in release 1.5 , we are able to migrate 7 repositories with their pipeline and finally we make sure they are compliant

Here is the status of all our repository.

Currently we have 10 repositories.

We have successfully migrated those pipelines we mark as green.

And currently we are working with those in pink color.

ndi-montiroing-app is one of the repo which we are working on.

As we notice some performance issue after we setup pipeline for that repo.

As Previously that ci pipeline is run in AWS and we are able to assign a 4 core codebuild for it. but in SHIP shared runner, they can only provide a 2 core container for each action which make our pipeline become very slow.

As efficiency is very important for a pipeline. Currently if we notice an issue in prod, we are able to release a patch from fix branch to dev within only half hour. But if we use new ci pipeline it will take one and half hour.

So we choose migrate that project after we able to adjust our pipeline runner’s performance

SRE doc and SRE development repo are waiting for the VPC setup.

**Here is the migration steps which we did for each repository.**

**Like create and config GitLab repo , setup rules**

**Migrate codebase from origin source.**

**Create and test ci pipeline and if all good we will**

Commit Gitlab ci template to old repo.

freeze code in old rep.

and then migrate to use new repo.

In this slide is to emphasize the difference flow of before and after migration. so   
we are going to use gitlab and gitlab ci instead of bamboo and bitbucket

And this flow explained how our currently ci pipelines looks like it is based on the original bamboo pipeline flow but we it split into different jobs and the pipeline

will be triggered based on changes of   
the feature and release branch.

The configuration of pipeline we aligned with IM8 policy and devsecops   
framework.

**And in release 1.7, we will migrate the outstanding repo to gitlab.**

**Create dashboard and do other enhancement.**

**and then signs off the migration completion with SHIP**

My name is Yihua, and I would like to share the achievements we have made in development framework, Epic.

In a previous release, we have built our development for infra , aws service and data access layer which enable us able to build more and more feature on top of those layer in release 1.6.

There are 4 features we build on top of data access layer .

First One is this query DSL which is our own domain-specific language base on python and used to query data from DynamoDB

And we have finish migrate all our lambda to use this approach.

Which means previously we must understand how dynamic db work and learn how to build the query to to fetch the data. But now we just need to know how to code in python as the API provided is straightforward compared with the one from aws.

This also means that our code has decoupled the dependency from AWS DynamoDB which will make it easier to migrate to other NoSQL solutions in the future as we just need to switch the implementation of our data access layer without need to touch the business logic.

Additionally, the API used by developers is generated directly from the documentation which eliminating the need for manually constructing SQL queries for our team members. This approach minimizes the risk of typos and syntax errors, as IDEs and pipelines can detect these issues through the generated code, which also enhances the overall development experience.

DynamoDB's performance is optimized when we follow its hash key and range key design pattern. However, our framework aims to eliminate this requirement technically. To address this issue,

We have developed a mechanism within our framework to detect if the query we build is a full scan or time long time to get result

This is already done in this release. and currently we manually run in before release but in a future release, we plan to build a monitoring system specifically for this purpose.

So it will ensure that all full scans and long-running queries are under control and closely monitored. This proactive approach allows us to take necessary steps before these issues escalate and cause significant problems.

Secondly, I would like to introduce Data Validation in our framework,

Data validation is a common requirement in application control and DevSecOps framework.

To address this for our data layer, we have seamlessly integrated the data validation process into the framework. And the validation is automatically applied and transparent to developers when framework is used .

By validating all incoming and outgoing data we can prevent the dirty data , unexpected behavior, and ensuring the integrity of our system.

Distributed Lock:

In our system, we encountered race conditions when multiple SNS triggers invoke same lambda function concurrently. Or 2 lambda operate the same resource at same time. And previously we write implementation base on each situation which makes code very hard to maintain and understand.

To address this issue and provide a standard solution.

We have implemented a distributed lock mechanism on top of our data access layer. We utilize the feature of unique constraint to acquire the lock which is very common for database.

By doing so, we can ensure that only one process gains access to a specific resource at a time, thereby preventing conflicts , maintaining data consistency and also our code become neat and easy to maintain.

Performance Monitoring:

As mentioned earlier, as there is a need to monitoring the query performance to prevent the abuse the data access layer.

We have recorded query, parameter, execution time for each our data access.

And this metric allow us to build the dashboard to get better understanding of the underlying system.

So we can easy know what’s the top query , Average time , count cross all our system..

And base on those information , we can know which query we should tuning and if we should do some optimize for our design easily.

Less code less bug

This may not 100% true, but each line of code could be a potential bug in future. And more code always means more testcase and logic to maintenance and it will long time to understand.

With the help of data access layer, we are able to reduce huge amount of code in our system. And compare with previous version ,in some lambda we can see it reduce 50% of code. And which means later we can have more time to build new feature instead of maintaining the old logic and do the bug fixing.

The 2nd layer we have implemented for development framework in this release is integration layer.

In this release , we convert all our lambda action to rest style API Definition. And we generate python code for team member to use directly from documentation which is same as what we did for the data access layer.

Of course , powered by the baseline we have setup for data access layer , we are able to convert all the lambda’s integration layer to use this approach with in only 16 points. And at the same time , it contains all the features and benefits like generated api, auto validation , decoupling dependence, monitoring , auto document publish. Reduce number of code. Which I think I won’t share again and again as it will be out standard cross all the layer.

Ok. So next step.

We Enhance DevSecOps pipeline for a comprehensive system protection.

And we will build release monitoring system in release 1.7.

we will use document first approach to build e2e test in release 1.8

And we will use document first approach to build controller layer in release 1.9

Thanks