Technical Onboarding - Quickstart guide

Introduction

Note: This information here assumes that you’re familiar with the Arkose Labs ecosystem at a high level. If you’re totally new, please take a moment to read and watch the following that will give you an introduction to the company and what we do:

- Arkose Labs Overview
- Arkose Labs Glossary
- EC-API PHP Architecture Overview
- Teams and Components

Welcome to the Enforce Services (#enfserv) team! We're primarily responsible for migrating away from the old monolithic PHP system into more performant Go based services. To date, some of the Go based services we've developed or are responsible for include:

- Setup Session
  - Responsible for establishing a users 'session' within the lifecycle of a request
- Store Analytics
  - Self-explanatory; Responsible for storing our analytics data for real-time and post-event analytics
- Verify
  - Responsible for providing value-added information to our customers about the authenticity of a session at its completion amongst other things

We're still currently working to migrate existing PHP services across to Go, so this list will be expanding and changing quite frequently. Alongside the migration work, we build new features, fix existing bugs, optimise for performance and reliability, and generally make sure that the Arkose system is providing the best experience possible.

Getting Started

Setting up your environment

The tools you'll use as a developer will change dependent upon your personal preferences, but the following is a good list of things which you'll probably need at one time or another. It's recommended that you read through the list in full before installing these, as there are a few different ways to go about doing that.

All OS's

- *Languages*
  - Required
    - Go v1.21 or newer
    - Node v18.17 or newer
  - Recommended
    - Typescript
- *Utilities*
  - Required
    - 1password
    - AWS Cli & AWS Vault
      - Setup guide can be located here
    - Direnv
    - Docker
    - Slack
    - Zoom
  - Recommended / Optional
    - ASDF
    - NVM
    - Postman / Insomnia
    - VSCode

OS Specific

- *MacOS*
  - Brew (recommended)
    - Many of the above languages and utilities will be available via `brew install x`

  - iTerm2 (optional)

The other tools that you use will be at your discretion. The team doesn't have a particular IDE that we use, however generally speaking most will opt for either VSCode or an IntelliJ variant (i.e. Goland).

Please note that administrative access to your local PC will be gated by the `Admin By Request` software. To perform any actions on your PC that would typically require administrative access (e.g. installing or updating software) you'll need to run that software and follow the prompts. Any issues can be directed to #it-support in Slack.

Environment hints and tips

There are a few things that you can do to avoid problems later on down the track

- When you have GitHub access, it's recommended that you create a Personal Access Token for read-only access and set that in your environment variables as `INSECURE_GITHUB_TOKEN`. This is sometimes used when building local images so that your build may pull in other private repositories (i.e. our shared library for Go services, go-lib)
- When you have NPM access, it's recommended that you create an Access Token for read-only access and set that in your environment variables as `NPM_TOKEN`. This will allow you to pull down our Organisation repositories during an `npm install`.
- Consider setting up the following environment variables:
  - `GOPRIVATE=github.com/ArkoseLabs/*`
  - `LOCAL_GOLIB_DIR=/path/to/your/go-lib`

Accessing our systems

Many of the applications and services we used are accessed through SSO via the Microsoft Apps page located here.

Requesting additional access

Many systems which you'll need for your day-to-day activities will require you to submit access requests through our IT support teams. As part of getting everything set up, it's recommended you check your access to the following systems and make requests where necessary:

- Github / AWS / ArgoCD / Jira / Confluence
  - These should be provisioned automatically as part of your on-boarding, however if not you may raise a ticket.
- Dataswan
  - Manages a variety of different services, however we mostly interface with it for access to Telltales (see the FAQ for more information on Telltales). Dev & Staging access should be fine to start with.
- NPM
  - Make an account with your company email before requesting an invite to the organisation
- SignalFX (Splunk)
  - APM & Metrics for our deployments.

Development Processes

Agile Scrum

The team operates by and large within the Agile Scrum framework, with a few minor differences. We run 2-week sprints and follow fairly typical ceremonies around this including Sprint Planning, Backlog Refinement, Daily Stand-ups, and a Retrospective at the end of the cycle.

One caveat to this is we don't however have a scrum master; as a group (manager & engineers) we're capable of fulfilling that role ourselves.

Git & Github

All of the repositories you will work with are under the companies Github account. Access to this will have been granted to you during the setup of your accounts with IT. At the time of writing, the main repositories you'll find us operating within include:

- EC-API
  - This mono-repository contains all of our re-written Go services, plus the original PHP code. It's expected we will migrate away from this pattern at some point in the future. repositories contains
- go-lib
  - This repository contains common components leveraged across multiple Go EC-API services. It does not compile into a binary, but rather acts as a library of components.
- Device Detection API (dd-api)
  - This repository primarily leverages an external PHP library to parse user-agents and value add to our fingerprinting processes

Tip: A nice and easy first PR is to go into our repositories and add yourself to the `CODEOWNERS` *file so that you'll be included in reviews*

Go

The ENFSERV team primarily develops our services in Go, and we take pride in writing performant, scalable, and well tested code.

For general guidelines and instructions on the approaches we take to writing code, including a few deviations from the communities 'norms', please see this document here. It's a great resource to understand how we structure our code, the conventions that we use, and some things we generally avoid. Of course, you're more than welcome to discuss any finer points with team in Slack or Zoom, but we've found this document to cover most scenarios that you'll encounter.

As you go through our services, you'll no doubt get a 'feel' for how we structure our projects. For the sake of consistency and ease of use, try to maintain that same style throughout your work as well. This isn't to say we can't make adjustments or improvements, but it's handy to keep all members working within our conventions and styles such that everyone can jump in and help and/or work with one another quickly and easily.

Code Reviews

For a short, sharp list of do's and don'ts for our code reviews, please see this document:

Our Internal Code Review Standards

To summarise the above document, we're responsible within our team for reviewing one another's code for quality, performance, and functionality. We're all professionals, but we're also human, so please be respectful during the review processes; we're all working towards the same goal.

Testing (Unit Tests, Integration Tests, E2E Tests, etc.)

The software that we produce within the team is amongst the most stringently and comprehensively tested code within the company. Every feature that we produce within our services should be accompanied by an appropriate amount of unit tests, integration tests, and e2e tests where appropriate.

Note: If you're unfamiliar with Go, or have had limited exposure to writing well covered tests for your features, it's recommended that you follow along with the `Learn Go With Tests` *resource available here. It's a great introduction into how you should structure and organise your tests. Beyond that, taking a look at any of our services and how we test would also be a good idea.*

Alongside the conventional Go unit tests, we also have a repository dedicated to standing up ephemeral testing environments where we run a series of Jest tests across our services to ensure there's no regressions between releases. You'll find that repository here, alongside a detailed explanation of how to get those up and running locally here. Its strongly recommended when working on creating new features, or modifying existing ones, to create supplementary API Tests to ensure that your feature works as expected for the current and future releases.

Releases

Whilst we're not fully CI/CD at this time, we do have a regular release schedule and aim to do deployments every other week for most services. Every member of the team is responsible for owning and executing releases on a cyclical bases (e.g. A → B → C → A → B etc. etc.) but don't fret as there's no expectation for you to take this on any time soon.

Enforce Services - FAQ's and Guides

How do I run the system locally?

That's a great question with a complicated answer. Regrettably, due in large part to the migration away from PHP to separate Go-based services, actually running the system from end-to-end can prove to be quite a difficult process (that will continue to be difficult until the migration is complete).

There is a detailed guide within EC-API that can assist you with standing up the entire system including both Go and PHP endpoints, but in our experience we've found that often times it's not quite necessary to stand up the entire system.

If you need to stand up a singular Go endpoint, a reasonable way to run a local copy of that service would be to leverage the API-Test repository. For this to work, you would need to build a local image from your development branch, and inject that into the API-Test startup process. The details of how to accomplish this aren't fully documented at this time, so please feel free to reach out to @Christopher Hollman on Slack for assistance.

How do I run our API tests and why would I want to?

The purpose of the API-Test framework thats been built out is to give us an ephemeral testing environment that we use, free from other users and teams, with a guaranteed system state. This means we can reliably and effectively test for regressions between releases, as those tests will pull down production images of those services for the tests.

The best resource for running those tests locally is the readme located within that repository here.

Alternatively, we also have workflows set up in EC-API that will automatically run API-Tests against your branch when required. This will save you the effort of having to do so manually!

What does our development process look like? How would I start a ticket?

The development process we employ is likely to be quite similar to other organisations you've worked at previously. Our administrative workflow is controlled via JIRA, where your tickets allocated to you will be visible from your dashboard. The moving of tickets between 'lanes' is self-explanatory; you just move them across as they progress through their lifecycle from backlog to done.

The actual code development process is also fairly straightforward. The generally recommended path would be:

1. Check out the repository which you'll be working in
2. Run `git pull` from the master branch to ensure you're working from the latest commits
3. Run `git checkout -b ENFSERV-XXXX/my-feature` to create and checkout a new branch for your work
4. Work on your feature, including:
   a. The feature itself; and
   b. Any associated unit tests
5. Validate your work will pass our automated checks with the shortcuts located within our `Makefile` including:
   a. `make vet` to run go vet against your code
   b. `make lint` to run `golangci-lint` against your code
   c. `make test` to run a full unit test suite against your code
6. Once ready, push your branch to remote and open up a PR
7. Fill out the PR with the necessary information to assist reviewers in understanding your code and open it for review
8. Ensure the necessary automated Github Actions and Workflows are passing, and remedy where necessary

9. Once you have the appropriate ticks, merging should be unlocked and you're free to merge your feature into `master`. If you don't have access, ask a Senior team member to do so for you.

When creating your PR, you'll also be prompted to consider other components to the development including any instructions for its release, adding additional tests to our API-Test repository, and more. Please sure you don't skip these prompts; they're important for code quality.

I need to add a feature that affects both our libraries (go-lib) and our services (EC-API). How do I coordinate that?

The generally recommended approach for this is:

1. Start your work where possible in `go-lib` first, building out that feature locally.
2. When it comes time to work in the service, you'll need to import the new `go-lib` locally. This is best accomplished by:
    a. Setting the `LOCAL_GOLIB_DIR` environment variable
    b. Running `make vendor-local-golib` within the service directory to import your local version of go-lib; and finally
    c. Running `make purge-local-golib` to restore the downloaded version of `go-lib` when no longer necessary
3. Once the local development of your feature is complete, the generally recommended order of operations is:
    a. Create a PR for your `go-lib` work
    b. Create a PR for your service work, but point your `go.mod` dependency to your remote branch:
        i. `GOPRIVATE=github.com/ArkoseLabs/* go get github.com/ArkoseLabs/go-lib@your-branch-name-goes-here`
    c. Once your `go-lib` work is merged, update your service PR to pull the latest `go-lib` version:
        i. `GOPRIVATE=github.com/ArkoseLabs/* go get github.com/ArkoseLabs/go-lib@master`

What environments are available to us?

Arkose generally runs three environments:

- Development
    - Able to be used for baseline testing of new features. The expectation is that this environment will generally be functional, but there is tolerance for downtime and regressions
- Staging
    - Able to be used to run pre-production testing prior to a production release. The expectation is that this environment will almost always be functional.
- Production
    - As expected, runs our live services and endpoints. Deployment to production is tightly controlled via ArgoCD and requires team members to go through the full release cycle process, coordinating with other team members and management.

Whilst there is some leeway as to how an individual wants to develop their features, we generally find that development on your local machine, supplemented with testing via the API-Test framework (either run manually locally, or automated when pushed to a remote branch with an open PR), is sufficient for most features.

How do I deploy my branch to development for testing?

Generally speaking, any changes that are merged into master will automatically be deployed to our deployed development environment. We do have provisions to manually deploy branches where necessary, however if you find yourself needing to do this it'd be best to coordinate that with a senior team member.

How do I deploy to staging / production? How do I run a release?

There's a comprehensive guide developed to walk you through the release process available here:
ENFSERV release admin process

I need to analyse a session / I need to view our logs / I need to query our analytics

We’ve got a really comprehensive set of reporting and analytics tooling here at Arkose, and there’s a few ways you’ll encounter to view this data. Here’s a non-comprehensive list:

Amazon Cloudwatch

In the first instance, it’s never a bad idea to use Amazon Cloudwatch to review the logs that our services will generate. It’s a really quick and easy way of consuming a lot of data, but is fairly raw and doesn’t offer much in the sense of analytics.

To access these logs, you would need to:

1. Navigate to MyApps
2. Click on Amazon SSO
3. Log in to the environment that contains the logs you’re searching for e.g. Arkose Production (Prod), Arkose Testing (UAT), or Arkose Development (Dev)
4. Navigate to Cloudwatch, ensuring that you’re in the correct ‘Region’ (top right of screen)
5. Run a query for the information that you need

Here’s an example query you could use to look for ‘error’ level logs in our Setup Session service:

```
fields @timestamp, kubernetes.namespace_name
| parse log '{"level":"*"' as level
| parse log '"msg":"*"' as msg
| parse log '"reason":"*"' as reason
| parse log '"session_id":"*"' as session_id
| sort @timestamp desc
| filter kubernetes.namespace_name = 'setup-session'
| filter log = 'error'
```

SuperSet

For a more robust analytics system, I’d recommend using Superset. This is a powerful tool we can use to amalgamate data, perform complex analytics of our system, create charts and graphs, and much more. Superset is also accessed via MyApps, but for a more detailed overview of this system I’d recommend following the Data team’s guide here: Superset User Guide

Splunk / SignalFX

When you need to look at our APM and metrics dashboards, Splunk is the go-to tool. Again, explaining the in’s and out’s of Splunk is beyond the quick hitting nature of this guide, but the following links will be useful to you:

- Splunk User Guide
- Enforce Services dashboards

General - FAQ’s and Guides

How can I get access to X? / Can I install Y? / I need technical assistance

Our IT Support team is available to help you with all of these issues. To start the process, please utilise the ticketing system here, or alternatively you can utilise the #it-support channel in Slack.

How can I see what the system looks like from start to finish?

To be able to run a ‘session’ from start to finish, it’s recommended that you take a look at our Test Cases website. This is a testing framework allowing any developer to simultaneously configure and run a session, whilst providing a lot of flexibility into exactly how it’s run. You’ll find options to run in dev/staging/prod environments, enable or disable certain features, and run the system in various customer based configurations.

What are ‘Telltales’?

Note: If you’re from Data / SOC and you have a better explanation, please feel free to edit this document and change this where you feel necessary. This is truly a barebones explanation for the purposes of getting new starters familiar with the term ‘Telltale’!

Telltales are what allow us to differentiate ourselves from our competitors; true to their namesake, they’re repeated patterns within traffic that we use to identify ‘bad’ actors.

Telltales are created either by our systems (AI / Computer generated) or by our SOC team (manual) for the purposes of identifying bad traffic so that we may take action against them (pressure). The Telltale itself is a rules based system, whereby we can leverage information we have on a session (what you’d consider their fingerprints) at any stage throughout the lifecycle of the request, validate whether that client matches any existing ‘rules’ (i.e. Telltales), and take actions based upon that.

This is a really, *really* simplified explanation. There is a ton of nuance surrounding Telltales that can’t be accurately summarised in a paragraph or two, so if you’re interested please look to the following document for more information on Telltales: Telltale System

What is this particular data field used for?

The Data team have set up a Data Dictionary that quite comprehensively goes over all of the fields you’re likely to encounter, and provides explanations on their origins and use cases. You’ll find that here:
Data Dictionary - From Dataswan - Verified Entries

Troubleshooting

How come I can’t load this service / Opening this website seems to load forever?

Some of our systems (e.g. ArgoCD, Splunk, Dataswan, etc.) are restricted to our internal network, so you’ll need to connect to the company VPN prior to loading them. Please follow this guide for more information.

Why can’t I pull the go-lib dependency down?

As `go-lib` is a private repository, you’ll need to slightly modify your command as follows (replacing master with your desired version or branch if necessary):

```
GOPRIVATE=github.com/ArkoseLabs/* go get github.com/ArkoseLabs/go-lib@master
```

I can’t seem to pull down images from our ECR?

You’ll typically encounter this problem trying to run API-Test’s trying to pull down production images from our Amazon ECR. To fix this, you’ll need to have successfully set up AWS Vault to get and set your SSO tokens. Assuming you have this fixed and working, do the following:

1. Run `aws-vault exec arkose-global`
   a. Ensure you login successfully via the prompted actions on AWS
2. Run the following command to export your access to Docker:
   a.
   ```
   aws ecr get-login-password --region ap-southeast-2 | \
       docker login \
       --username AWS \
       --password-stdin "${AWS_ECR_ACCOUNT_ID}.dkr.ecr.${ECR_REGION}.amazonaws.com"
   ```