



Onboarding

Onboarding New Members

Here's a checklist of things new team members need access to.

- ☐ Slack
 - ☐ Ask Deb
- ☐ Github
 - ☐ Ask Prarthna
- ☐ Notion
 - ☐ Ask Deb
- ☐ Figma
 - ☐ Ask Vidushi
- ☐ Google/Gmail (If getting techjapan.work email)
 - ☐ Ask Masahiro Shinotsuka
- ☐ JIRA / Notion - For tracking issues/tickets
 - ☐ Ask Deb
- ☐ AWS
 - ☐ Ask Deb
- ☐ Hub Account
 - ☐ Admin
 - ☐ Student
 - ☐ Company

☐ CA

☐ Ask Deb

Github Repositories

All our codebase is hosted under TechJapan organization on Github. Here are some key code repos you might need to look into.

1. Tech-Japan/hub - The main codebase for Hub platform
2. Tech-Japan/joblink - Linkedin extension codebase
3. Tech-Japan/TechJapan.work - Codebase for the landing site (techjapan.work)

Hub Environments

We had 3 kinds of environments (or apps) for the hub platform

1. hub.techjapan.work - The main production site. It is mapped to the `master` branch on Github. If you push to that branch this site gets updated.
2. staging.techjapan.work - Staging or QA site. This has the exact same codebase as the main site because this also gets deployed from `master` branch automatically. This can be used as a testbed to try out actions and features on the site without affecting real production data. The database used for this server is a snapshot of the main production database so it will look and feel like the real one. Note that the data could get stale over time (as it's only a snapshot) so you could refresh by re-copying over the data from time to time.
3. pr12.hub.techjapan.work - PR preview sites. These look like `prXXXX.hub.techjapan.work` where XXXX could be the your pull request number from Github. These sites are auto-deployed anytime you create a Pull-Request on Github. These are sandbox environment used to test out the feature that PR is implementing. They get deleted when the PR is closed. They are completely isolated and self-contained. They have their own local database and Redis instance with test dummy data.



All PR preview environments have the same following credentials.

Email:

`demo@email.com`

Password:

`password`

AWS

We use AWS for most of our infrastructure. As a result it's important for you to familiarize with AWS console. Important to note that we use `ap-northeast-1` as the main region of deployment for most of the resources. Some AWS resources are 'global' and might be in `us-east-1` region. Here are some key locations that you might need.

1. IAM: use it to create your local access keys and create/manage users
2. S3: Files and buckets
3. Loadbalancer: We have an HTTP loadbalancer to route requests to our app servers
4. ECS: We use ECS for app deployments. All our apps run as docker containers
5. ECR: to deploy a docker app you must first upload a docker container to registry. This is it.
6. Secrets Manager: Stores all our secrets for each environment
7. Route53: DNS configurations
8. Cloudfront: CDN for delivering fast static contents such as files, resumes, and JS/CSS content
9. ACM: To serve HTTPS traffic we need SSL certificates. This is where we manage those
10. RDS: SQL databases
11. SES: Email service for sending emails from our apps
12. Elasticache: Redis cluster for use as a cache in our apps

Secrets

Each environment needs different configuration to operate, for eg: database connection password, cron tokens etc. Most of these are sensitive info and should not be stored in the codebase.

We use AWS Secrets Manager to host all the secrets for each environment, Our automated build pipeline automatically uses the correct secrets for the correct environment. In case you need to see the actual secret values, [you can view all the hosted secrets here](#)

1. `techjapan-hub-production` - Main production secrets
2. `techjapan-hub-staging` - Staging env secrets
3. `techjapan-hub-preview` - All PR preview envs use this one

Databases

We use RDS to host our mysql databases. We have one [RDS instance that can be found here](#) that contains several databases listed below

1. techjapan-hub - This is the main production database. Handle with CARE
2. techjapan-hub-staging - This is the copy of main production database used for `staging.techjapan.work` site.
3. tj-landing - Used for the headless wordpress for the main landing site



NOTE: By default, the database access is restricted to only inside AWS private network. This means if you try to connect to it via your local IP it won't work. You can temporarily allow access from AWS security group settings ([Link here](#)) . Just add a new 'Inbound Rule' to allow your IP on port 3306

Database Migrations and Seed

Find db related files under `db-data` folder

File Buckets

Hub needs to store lots of files for proper functioning. Candidates can upload their resumes, companies can upload job descriptions etc.

We use S3 to host all such files. These are the buckets that host files for our environments

1. `techjapan-hub-files` - Main production bucket.
2. `techjapan-hub-files-staging` - Files for the staging server
3. `techjapan-hub-files-prXXX` - Files for a particular Pull request on Github. These get auto-deleted when the PR is closed

CDN

We serve all our static content via CDN to speed up delivery and improve caching. Here is the link

[CloudFront \(amazon.com\)](#).

Redis Cache

Redis server can be used in various ways like ratelimiting, caching etc. Using redis as a cache can also reduce load on our databases. [You can view our redis cluster here](#).

Here are some related code files

- [hub/redis.js at master · Tech-Japan/hub \(github.com\)](#).
- [hub/cache.js at master · Tech-Japan/hub \(github.com\)](#).

Email

We use SES to send emails. [The SES dashboard can give you most of the details](#)

On the application side: we configure `nodemailer` to send emails via SMTP protocol via SES. [Here is the related code](#). The SMTP credentials can be generated from the SES dashboard.

Cron Jobs

There are several “jobs” that need to be run on a schedule for Hub. For eg. generating recommendations for internships, cleaning old user data etc. These are done via cron jobs.

All the cron jobs are also declared in the code base. We use “HTTP” endpoint based job design which makes operating crons jobs simpler (because each cron job is just another endpoint in our app). To create a new CRON job we first create an API endpoint (like `/api/crons/do-something`) and then setup schedules via AWS Eventbridge to hit that API periodically.

The code to declare new jobs is

[here](#) and you can see some of our existing cron job endpoints [here](#)

Because we use an HTTP endpoint for crons, any random hacker could call those API endpoints if they know the path, which wouldn't be very safe. To prevent this, we require that caller must pass a secret token when calling the CRON api `?token=XXXXX` parameter. This is handled automatically by our code deployment which puts a token when calling the endpoints, but if you are calling those endpoints yourself then you must pass the right token. The correct token to pass can be found in the environments Secret

Infrastructure as Code

We don't deploy any of our resources (such as app, database, dns, buckets) manually by hand. Instead we use AWS CDK to create these in a repeatable consistent way.

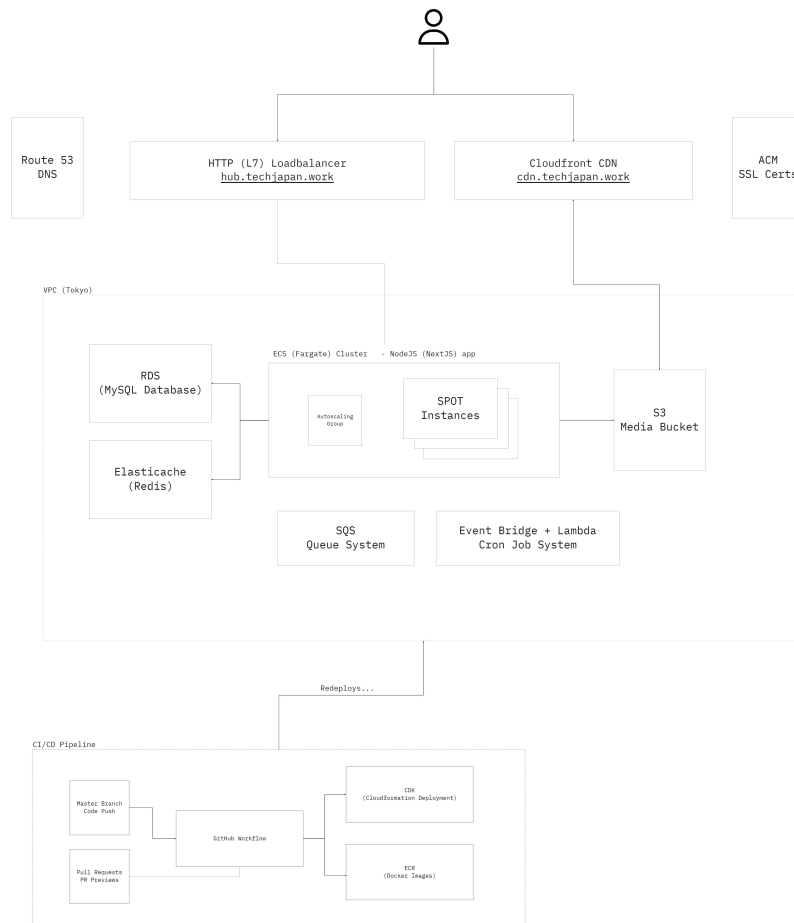
All the code for these deployments can be found at `cdk` folder in the codebase [Link here](#)

CDK deploys our resources by using AWS Stacks. [You can view all our stacks here](#)

[class CfnDBCluster \(construct\) · AWS CDK \(amazon.com\)](#)

App System Design

Here's how our app deployment infrastructure looks like



CDK Deployment via CLI

```
cdk diff techjapan-hub-pr0
cdk deploy techjapan-hub-pr0
cdk destroy techjapan-hub-pr0
```

CI/CD Pipelines

We use Github Actions as continuous integration tool. We have 3 pipelines at the moment

These are declared here

1. `on-master-push` - This gets triggered when we push to master branch or merge a PR into the master branch. This lints, builds, and deploys the codebase to AWS via CDK
2. `on-pull-request` - This gets triggers when someone creates a new pull-request. This auto-deploys a PR-preview sandbox deployment
3. `on-pr-closed` - This triggers when you merge or close a PR. This cleans up the PR preview environment and all related resources.

How we work on new features

1. After discussion the nitty gritty details for a feature with the team, we make code changes in a new branch and create a pull request from that. For the branch name we have been using `developer/feature` format, for eg: `kashif/new-list-candidates-api`
2. Always make sure to create branches from the latest master by pulling master code frequently
3. Once the PR is ready, request a review from one or two relevant team members.
4. Make sure to get at least one approval on your PR before merging
5. Once you have approval, you can merge the PR yourself.

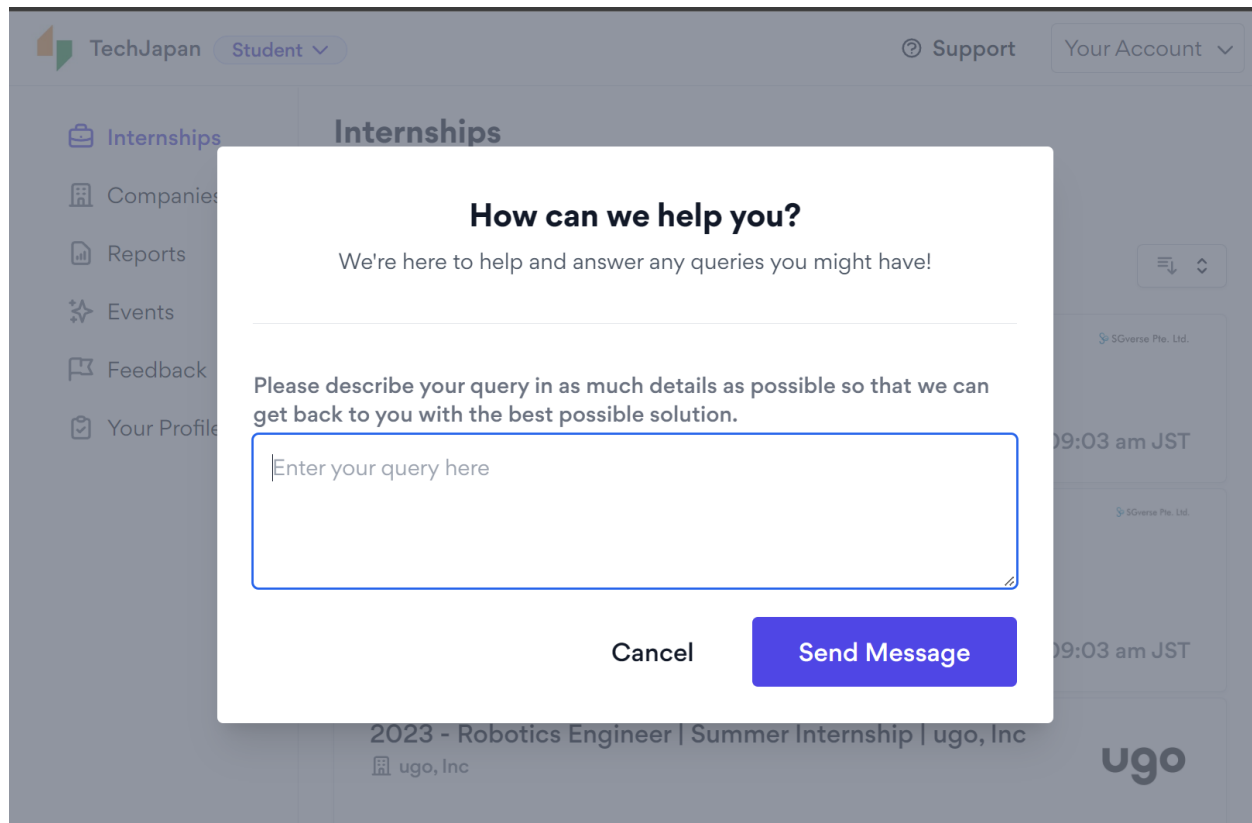
Support Requests

We don't have a dedicated support helpdesk system for the Hub at the moment. But this could change in future as we start to deal with a lot of incoming support requests.

For now we have a simple button on the platform that users can use to send us support queries. Submitting text on this sends a message directly to our `hub-`

`support-requests` Slack channel ([Sample here](#)).

Later @Deb picks them up and resolves queries manually.



Error logs and Notifications

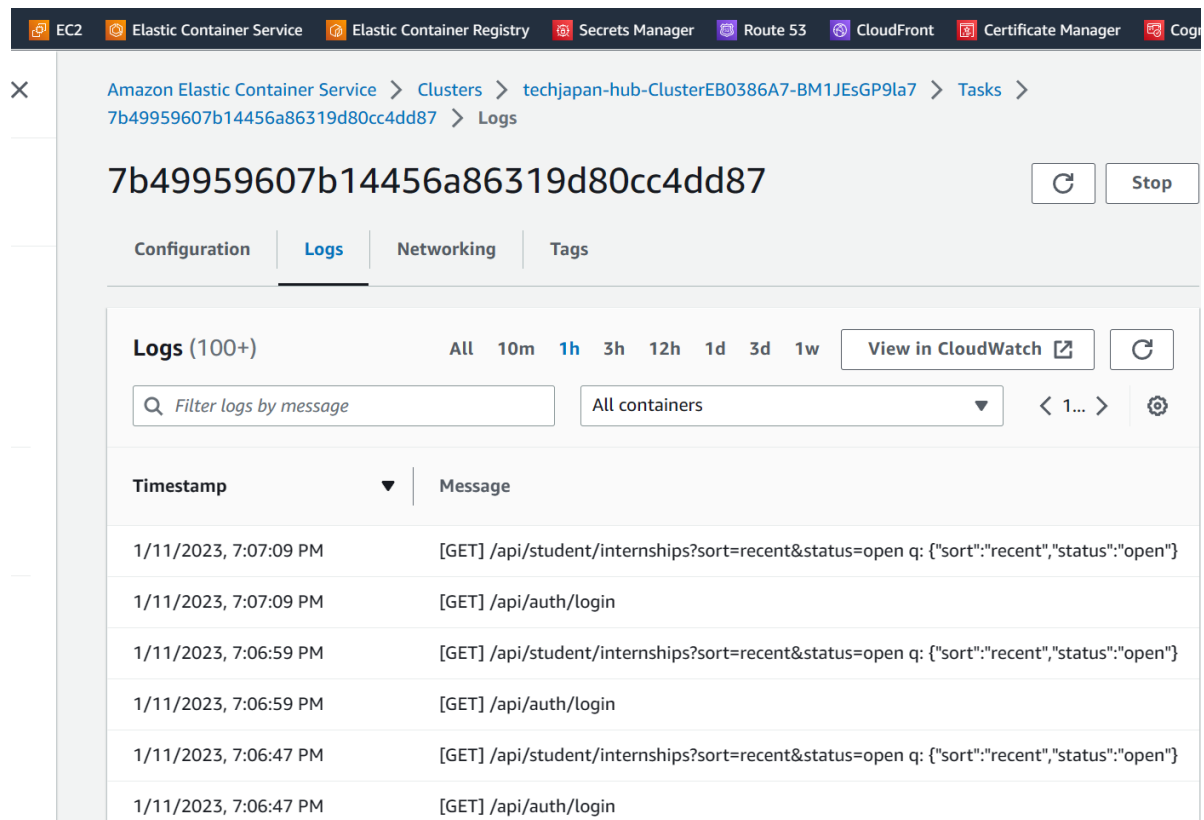
Any error log generated in the app (either via `console.error()` command or due to exceptions) are sent to `hub-notifications` channel on Slack ([Sample here](#)). While this isn't as robust as having a dedicated error/log monitoring system, it's been quite handy so far to figure out bugs in our app quickly.

You can see how we hook up the logger to send error logs to our Slack here

- [hub/server.js at master · Tech-Japan/hub \(github.com\)](#)
- [hub/logger.js at master · Tech-Japan/hub \(github.com\)](#)

If you would like to see the realtime logs for running applications, you can do so from AWS console under ECS tasks.

▼ Here is an example of our main hub app: [Amazon ECS](#) . Find the active running task and their view logs under 'Logs' section



The screenshot shows the AWS Management Console interface for the Amazon Elastic Container Service (ECS). The breadcrumb navigation indicates the path: Amazon Elastic Container Service > Clusters > techjapan-hub-ClusterEB0386A7-BM1JEsGP9la7 > Tasks > 7b49959607b14456a86319d80cc4dd87 > Logs. The task ID 7b49959607b14456a86319d80cc4dd87 is prominently displayed, along with 'Refresh' and 'Stop' buttons. Below this, tabs for 'Configuration', 'Logs', 'Networking', and 'Tags' are visible, with 'Logs' being the active tab. The 'Logs' section shows a list of 100+ logs, with filters for 'All', '10m', '1h', '3h', '12h', '1d', '3d', and '1w'. A search bar and a dropdown for 'All containers' are also present. The log entries are displayed in a table with columns for 'Timestamp' and 'Message'.

Timestamp	Message
1/11/2023, 7:07:09 PM	[GET] /api/student/internships?sort=recent&status=open q: {"sort":"recent","status":"open"}
1/11/2023, 7:07:09 PM	[GET] /api/auth/login
1/11/2023, 7:06:59 PM	[GET] /api/student/internships?sort=recent&status=open q: {"sort":"recent","status":"open"}
1/11/2023, 7:06:59 PM	[GET] /api/auth/login
1/11/2023, 7:06:47 PM	[GET] /api/student/internships?sort=recent&status=open q: {"sort":"recent","status":"open"}
1/11/2023, 7:06:47 PM	[GET] /api/auth/login

00_Dev Onboarding (ARCHIVED)