Roadmap: Riju architecture improvements

This roadmap lists a sequence of small changes that can be made to improve Riju's architecture. The goal is for each change to be implementable separately from the others, and to immediately yield an improvement even if no further work is done.

Prometheus metrics

What: Instrument the application server with Prometheus metrics such as resource utilization, container count/age by language, connection interrupts, etc. and send them to Grafana Cloud using Grafana Agent.

Why: Fix the observability hole where it's difficult to tell why Riju is under load when it pages.

Static assets

What: Serve static assets for Riju from S3 rather than from the API server directly.

Why: This allows changing the frontend without having to redeploy the API server, improves frontend deployment times significantly, and also ensures that a proper error message can be displayed to users when the API server is unavailable.

Riju on Kubernetes

What: Set up a new node AMI that has k3s installed, and replace the supervisor binary with a Kubernetes deployment manifest. Instead of an ALB together with supervisor blue/green deployments, use ingress-nginx-controller for load balancing and cutovers within the cluster, and use DNS cutover to perform whole-cluster replacement.

Why: The ALB is 15% of Riju's monthly spend; replacing its budget with additional compute will increase scalability. Additionally, replacing the supervisor binary with Kubernetes native alternatives will improve reliability by using production-ready third-party container orchestration code instead of hand-rolled Go. Finally, introducing Kubernetes to the architecture unlocks a number of future improvements in reliability, scalability, and simplicity.

Self-hosted container registry

What: Replace the ECR repository with a self-hosted Docker registry instance backed by S3 and running as part of the Kubernetes cluster.

Why: ECR is 4% of Riju's monthly spend; replacing its budget with additional compute will increase scalability. Additionally, hosting our own registry will give us additional control over access control and retention policies which are not customizable in ECR. This customizability will make it easier to implement features down the road for accessibility to contributors (e.g. ability to pull existing Riju base images instead of building them from scratch). Also, running more services in the cluster will help to surface any deployment complexities with the new environment early, before migrating more critical components of the architecture.

Kubernetes native orchestration

What: Create user sessions as pods via the Kubernetes API instead of using Docker directly.

Why: Using production-ready Kubernetes primitives for container orchestration will be much more reliable than hand-rolled Docker commands and C code, which have suffered repeatedly from resource leaks and general flakiness.

Server agent binary

What: Separate out the part of the webserver that deals with translating user session API commands into container process invocations, and automatically publish versioned tarballs of it. At runtime, download, extract, and cache this tarball, mount it read-only into the user session container, and run it inside the container to respond to websocket requests. Store the tarball version for each language in a JSON file in S3, and create some basic command-line tools to perform common operations such as updating one or all languages to the latest version. Merge that file into the deployment configuration file that is read by the supervisor process.

Why: The goal is to identify the specific subset of the Node.js code that could break per-language tests if changed, and to allow different languages to use different versions of that code. This means that any part of the Node.js code can be changed without having to re-run all language tests, greatly increasing development velocity. In addition, adding the server agent binary to the architecture means that it is much more obvious when tests need to be run (tests for a language only need to be run when that specific language is explicitly modified), greatly decreasing the need for a complex modification detection algorithm for dependency tracking.

Stateful dependency management

What: Remove the content-hash-based dependency tracking system currently used by Riju. Replace it with a Postgres database (self-hosted in Kubernetes, backed by EBS) that tracks all compiled language artifacts, and update command-line tooling for artifact management to also manipulate corresponding database entries. The database will be queryable manually to answer questions such as: What languages have not been updated recently? What configuration was used to compile a particular language? Which languages use an old version of Ubuntu in their base image? What specific combinations of language configuration, base image, and server agent have been tested together successfully?

Why: This is the second phase of eliminating Riju's lengthy compilation process. With the server agent binary component in place, it will be safe to make arbitrary modifications to Riju's code without having to re-run any tests or rebuild any artifacts other than the ones desired for the specific language(s) changed. This means the slow, complex, and error-prone Depgraph component of Riju can be entirely eliminated. Replacing Depgraph with a relational database will not only improve velocity, but also increase visibility into the current deployment state, since the entire base configuration will be available rather than just its content hash. Furthermore, moving to a stateful dependency model is a requirement for supporting in-place language updates submitted by the community or generated by an automated build system. Finally, having the language artifacts backed by a stateful database makes it straightforward to build and deploy only a subset of languages if a third party wishes to operate their own instance of Riju.

Relax resource constraints

What: Scale up the maximum allowed resource consumption for user sessions considerably.

Why: With a robust Kubernetes-based container orchestration system, it should be possible for Riju to be fairly robust to misbehaving user sessions. Increasing the allowed resource consumption would allow tests for several slow languages to pass and thus for those languages to be re-added.

Remote build system

What: Add an authenticated endpoint that allows triggering artifact compilation (e.g. Debian package or Docker image) within the cluster and storing the result in the shared database and object store. Allow offline log retrieval from build jobs as well as streaming to multiple clients over websocket. Use Kubernetes native pods as build runners that share resources with the rest of the cluster.

Why: Bandwidth constraints make it inconvenient to rebuild and upload many languages locally; in addition, the local workflow is not accessible to external contributors. Moving to a hosted build system makes it once again possible to rebuild a large number of languages without significant manual work, and opens the doors to future work to improve accessibility of Riju to contributors.

Developer dashboard

What: Create a public dashboard in the Riju web interface that shows the current deployment state of each language: when it was last built, what version of Ubuntu and the base image are used, which server agent binary, and all relevant language configuration YAML.

Why: This makes it possible for external contributors to see the state of what's deployed in Riju, which is a first step towards first-class support for such contributors in the architecture.

Sandbox environment

What: Create an advanced version of the normal Riju web interface that allows selecting an arbitrary combination of Docker image, language configuration, and server agent version.

Why: This allows for rapid testing of simple configuration changes, and additionally opens the door to testing external contributions in a publicly auditable way.

Public image registry

What: Set an access control policy (or create a proxy) for the image registry that will allow a subset of Docker images to be pulled by clients outside the cluster.

Why: This will allow local development by external contributors using the same environment as will be used in production when their contributions merge.

GitOps for dependency management

What: Create a YAML format for *change requests*, which are files that specify a set of languages to be updated, and the stage from which to re-start their compilation (e.g. language configuration, image build, Debian package build). Use GitHub Actions to provide a text-based interface in pull request comment threads to ask that a change request be built. Add an approval step for first-time contributors, and expose the state of the change request build in the developer dashboard for all pull requests on the main repository. Provide an administrator endpoint for promoting the built artifacts of a change request to the live environment.

Why: This makes it possible for external contributors to propose new languages and language updates in a way that isn't prohibitively difficult to review, massively decreasing iteration time.

Streamlined local development

What: Refactor shell scripts and Makefiles to make them more foolproof for local development, and remove the need to run nested tmux sessions.

Why: Local development is necessary when adding a new language or seriously modifying an existing one, but it is complex and cumbersome, which is a barrier for new contributors.

Automated language updates

What: Create a cron job that will automatically attempt to use the remote build system to rebuild languages that have not been updated in a while. The cron job will create a CR automatically, upon which internal or external contributors can build in order to diagnose any failing language updates. Flag failing languages so that another update is not attempted until they are modified to fix the issue.

Why: This would eliminate the massive amount of manaul maintenance that would be required to keep languages up to date (or even ever updated) on Riju, which currently doesn't happen at all.