Linux Job Worker

Summary

Implement a prototype job worker service that provides an API to run arbitrary Linux processes.

Rationale

This exercise has two goals:

- It helps us to understand what to expect from you as a developer, how do you write production code, how you reason about API design and how you communicate when trying to understand a problem before you solve it.
- It helps you get a feel for what it would be like to work at Gravitational, as this exercise aims to simulate our day-as-usual and expose you to the type of work we're doing here.

We believe this technique is not only better, but also is more fun compared to whiteboard/quiz interviews so common in the industry.

We're not alone:

http://sockpuppet.org/blog/2015/03/06/the-hiring-post/

We appreciate your time and looking forward to hack on this little project together.

Objective

The goal is to implement a small part of a distributed job scheduler: a linux job worker server that executes arbitrary linux processes based on the direct API requests from clients.

Requirements

API

- Job worker should provide RPC API to start/stop/query status and get an output of a running job process. Any RPC mechanism that works for the task and is familiar to you is OK: GRPC, HTTPS/JSON API or anything else that can guarantee secure and reliable client-server communication.
- The API should provide simple but secure authentication and authorization mechanism.

Client

- Client command should be able to connect to worker service and schedule several jobs
- Client should be able to query result of the job execution and fetch the logs

Guidance

Interview process

The interview team is assembled in the slack channel and consists of the engineers who will be working with you, you are encouraged to chat to them and ask questions about the engineering culture, work and life balance, or anything else that you would like to learn about Gravitational.

We understand that the interview is a two-sided process and we'd be happy to answer any questions!

Before writing the actual code, we encourage you to create a small design document in Google Doc and share with the team. This document should consist of key trade-offs and key design approaches. Please avoid writing overly detailed design document, use this document to make sure the team could provide a design feedback and demonstrate that you've investigated the problem space to provide a reasonable design.

Split your code submission using pull requests and give the team an opportunity to review the PRs. The good *“rule of thumb”* to follow is that the final PR submission is a formality adding a small feature set - it means that the team had an opportunity to contribute the feedback during multiple well defined stages of your work.

Our team will do their best to provide a high quality review of the submitted pull requests in a reasonable time frame. You are spending your time on this, we are going to contribute our time too.

After the final submission, the interview team will assemble and vote using +1, -2 anonymous voting system: +1 is submitted whenever a team member accepts the submission, -2 otherwise.

In case of positive result, we will connect you to our HR team who will collect one-two references and will work other details. You can start reference collection process in parallel if you would like to speed up the process.

After reference collection, our ops team will send you an offer.

In case of a negative score result, hiring manager will contact you and send a list of the key observations from the team that affected the result. Please don't be discouraged, our code review process is focused on the submission, not the candidate and we will be excited for you to take another challenge at a later time if you feel you've addressed our comments!

Code and project ownership

This is a test challenge and we have no intent of using the code you've submitted in production. This is your work, and you are free to do whatever you feel reasonable to it. In the scenario when you don't pass, you can and should open source it with any license and use it as a portfolio or anything else.

Areas of focus

Gravitational focuses on networking, infrastructure and security, that's why these are the areas we will be evaluating in the submission:

- Consistent coding style. Gravitational follows https://github.com/golang/go/wiki/CodeReviewComments for the Go language. If you are going to use a different language, please pick coding style guidelines and let us know what they are.
- Please write one test for authentication and another for the networking component.
- Reproducible builds. Pick any vendoring/packaging system that will allow us to get consistent build results.

- Consistent error handling and error reporting. The system should report clear errors and do not crash under non-critical conditions.
- Concurrency and networking errors. Most of the issues we've seen in production are related to data races, networking error handling or goroutine leaks, so we will be looking for those errors in your code.
- Security. Use strong authentication and simplest, but robust authentication. Set up the strongest transport encryption you can. Test it.
- Error logging and handling. Consistent errors are the key

Trade-offs

It is important to write as little code as possible, otherwise this task could consume too much time and overall code quality will suffer.

It is OK and expected if you cut corners, for example configuration tends to take a lot of time, and is not important for this task, so we encourage candidates to use hard codes as much as possible and simply add TODO items showing candidate's thinking, for example:

```
// TODO: Add configuration system. Consider using CLI library to support
both
// environment variables and reasonable default values,
// for example https://github.com/alecthomas/kingpin
```

Comments like this one are really helpful to us, because they save yourself a lot of time and demonstrate that you've spent time thinking about this problem and provide a clear path to solution.

Consider making other reasonable trade-offs, make sure you communicate them to the interview team. Here are some other trade-offs that will help you to spend less time on the task:

- Do not implement a system that scales or is highly performing. What performance improvements would you add in the future?
- High availability. It is OK if the system is not highly available. Write down how would you make the system highly available and why your system is not.
- Do not try to achieve full test coverage. This will take too long. Take two key components, e.g. authentication/authorization layer and networking and implement one-two test cases that demonstrate your approach to testing.

Pitfalls and Gotchas

To help you out, we've composed a list of things that resulted in a no-pass from the interview team:

- Scope creep. Candidates have tried to implement too much and ran out of time, energy. To avoid this pitfall, use the simplest solution that will work. Avoid writing too much code. We've seen candidate's code introducing caching and making many mistakes in the caching layer validation logic. Not having caching would have solved this problem.
- Data races. We will scan the code with race detector and do our best to find data races in the code. Avoid global state as much as possible, if using global state, write down a good description why is it necessary and protect it against data races.
- Deadlocks. When using mutexes, channels or any other synchronization primitives, make sure the system won't deadlock. We've seen candidate's code holding mutex and making a network call without timeouts in place. Be extra careful with networking and sync primitives.
- Unstructured code. We've seen candidates leaving commented chunks of code, having one large file with all the code, not having code structure at all.
- Not communicating. Candidates who submitted all the code to master branch, which does not give us the ability to provide feedback on the various implementation phases. Because we are a distributed team, structured communication is critical to us.
- Implementing custom security algorithms/authentication schemes is usually always a bad idea unless you are a trained security researcher/engineer and definitely a bad idea for this task - try to stick to industry proven security methods as much as possible.

Scoring

We want to be as transparent as possible on how we will be scoring your submission. The following table provides a description of different areas you will be evaluated on and how they will affect your overall score.

Description	Possible Points Awarded	Possible Points Subtracted
The submitted code has a clear and modular structure	+1	-1
The candidate communicated their progress during the interview	+1	-1

The program builds are reproducible	+1	-1
README provides clear instructions	+1	-1
The candidate outlined the key design points in the design document	+1	-1
The code has no obvious data races and deadlocks	+1	-1
The code provides examples of tests covering key components	+1	-1
The code provides clear error handling and error reporting	+1	-1
The program is working according to the specification	+1	
The candidate demonstrates ability to handle and apply feedback	+1	-1
The Client-server communication is implemented in a secure way	+1	-1

Asking questions

It is OK (and encouraged) to ask the interview team questions. Some folks stay away from asking questions to avoid appearing less experienced, so we provide examples of questions to ask and questions we expect candidates to figure out on their own.

This is a great question to ask:

> *Is it OK to pre-generate secret data and put the secrets in the repository for the purposes of POC? I will add a note that we will auto-generate secrets in the future.*

It demonstrates that you thought about this problem domain, recognize the trade off and save you and the team time by not implementing it.

This is the question we expect candidates to figure out on their own:

> *What version of Go should I use? What dependency manager should I use?*

We expect candidates to be able to find solutions to common non-project specific questions like this one on their own. Unless specified in the requirements, pick the solution that works

best for you.

Tools

This task should be implemented in Go and should work on 64-bit Linux machine with kernel greater than 3.19.0.

Communication

When in doubt doubt, always err on the side of over communicating - it’s better to ask, we promise we are not going to subtract any points for seemingly “silly” questions.

And THANK YOU for taking your time and taking on the challenge. We understand that your time is valuable and we really appreciate it!

We wish you good luck!