SHARED PUBLICLY

PRR <FeatureName>

Author(s)	Who wrote this
Start Date	YYYY-MM-DD
End Date	YYYY-MM-DD
Status	**Not Started, In Progress, Done**
Approver(s)	Who signs off on the PRR and is your guide?
Informed	Who is important to keep informed about this PRR?
Product DNA	Link to Product DNA doc if this is a customer facing feature
Design Doc(s)	Link to one or more design docs
Delivery Plan	Link to the Delivery Plan for this project

Instructions:

- Make a copy of this checklist
- Use the spaces below to fill in answers to the questions.
- This is expected to take a day or so, assuming you have good answers. Much longer if you don't...
- Provide links to supporting material (design docs, licenses etc)
- The questions are meant to highlight areas we may have missed.
- The questions are more guidelines than hard-and-fast rules.

Architecture & Scaling

Is the service's architecture documented?
Original design documents suffice. Ensure they are placed in the Design Docs folder. Extra credits go for a training session on the new service. Link to supporting material below.

Can the service tolerate machine failures whilst preserving SLO?
All services need to be able to tolerate at least a single machine failure without user-visible impact. Relying on Kubernetes to restart your Pod on another node is not usually sufficient.

SHARED PUBLICLY

Are there any single points of failure? Examples include central databases, dependencies on external services etc

Are the service components easily scalable?
Ideally, we want to design services which can scale to an order-of-magnitude (i.e. 10x) of their estimated initial load. In most cases, a stateless service behind a load balancer is expected. For a service made up of multiple microservices, please document the scaling approach for each one. *Consider if there are any contended resources that might cause slowdown? Examples include central databases, locks or quotas on external services.*

Can users consume unbounded resources through your service?
We expect services to impose limits on user behaviour, from a self-preservation perspective. Eg users shouldn't be able to upload 1GB of alert rules. Inputs should be bounds checked.

Do you use any third-party non-managed dependencies?
Examples include Cassandra, Elasticsearch, Consul, etcd, MySQL. We strive to use managed services where possible. If you use a non-managed, third party service, please consider if there are enough people with operational experience that can take in escalations and resolve production issues? Is there a mechanism to easily scale the external dependencies? (eg bigger clusters for bigtable, sharding for SQL databases)

SHARED PUBLICLY

Code, CI, Releases

Code

Where is the code hosted?
The only acceptable location is our organisation on Github. *Directly using upstream OSS projects is fine, but if running from a fork is needed, have it in our Github org rather than your own GH account.* *Ask yourself if the repository is correctly configured - is the master/main branch protected? Who can write to the repo? Is the repo private (where appropriate)?*

If this is a our company's OSS project, does it have a CLA setup, is there an appropriate license and governance doc?
We intentionally prefer using a CLA over a DCO for our OSS projects. We use AGPLv3 for our main projects, and Apache2 for tools, libraries, devkits, and such.Any exception should be justified. We want all of our first-party projects to use the same governance.

CI & Testing

If you are using upstream built assets for established OSS projects, this section doesn't apply.

Do you have CI? Do you use an existing "blessed" CI solution?
All our projects should have CI configured to run at least unit tests and linting. *AA, BB and CC are considered "blessed", with AA being the default choice.*

Does the CI build the final production assets? Where are the built assets stored? Are they versioned?

SHARED PUBLICLY

We expect all assets run in production to be built by CI, not on a developers laptop. *Acceptable answers are: our Docker hub account, cloud provider's private docker hub, a bucket for apps & plugins…* *We expect all assets to be versioned; use the git sha as the label, no pushing to :latest docker tag.*

Release Process

Is the release process documented? How regularly do you release?
You should document the process of getting a new build of your service into production; putting these docs in the infrastructure configuration repository is a good idea. *We encourage teams to release often, once a week is recommended as a minimum, even if there are no changes worth deploying.*

Is there a staging environment? A dev environment?
Don't push directly to prod. Have extra environments where you "stage" changes first. Staging should have a representative workload on it. *The dev environment is separate from the staging environment. It's a playground, free to use for all developers of the service.*

Can updates to the service be rolled out without downtime? Can releases be safely rolled back?
Updates to the service should not impact an unreasonably large percentage of the system at once. Updating infrastructure configuration shouldn't result in downtime. Give thought to schema migrations on rollout / rollback, we don't want partial/invalid data being served.

SHARED PUBLICLY

Config Management

Is the service config version controlled? Is the config in the infrastructure configuration repo?
The config for all jobs running on our Kubernetes clusters should be version controlled in the infrastructure configuration repo. *All config should use Jsonnet and Tanka to configure deployments. The config should be in a library and shared between environments, to ensure consistency of deployment.*

Are resource requests and limits appropriately configured?
This is a complicated topic and should be thoroughly discussed with the PRR guide. *In very general, requests should be set to the 95th quantile of the usage over the last week.*

Are your jobs restarted when config changes?
The most straight-forward way is to prefer command line flags over config files. *If you use config files, you need to arrange for your jobs to restart when the config changes eg using annotations of the config hash.* *We do not recommend having dynamic config in most situations, but upstream projects might require it (eg Prometheus).*

If you have external dependencies, are they configured with Terraform?
eg S3 buckets, RDS instance etc

SHARED PUBLICLY

If the service required secrets, are they in an approved reliable secret store?
e.g. Vault, Cloud KSM, Secrets Manager, etc.

Security

How is access to your new service controlled?
Internal services should only be accessed via the authenticated endpoints. *Customer services should use an access token or oauth.* *Kubectl port-forward is not acceptable.*

Is TLS used over all untrusted networks?
E.g. anything that goes between the customer and our Kubernetes clusters, or our Kubernetes clusters and third party services, should use TLS. *Our internal (pod-to-pod) network is trusted, as in our intra-cluster network - you don't need to use TLS on these.* *Services operated by us must receive an 'A' grade or higher when tested on Qualys' "SSL Server Test" service. This will be ensured if using a global load balancer or k8s Ingress.*

Is sensitive data encrypted at rest?
This is ensured when using anything in the cloud platform; passwords and other credentials should not be stored in clear text, not even on storage managed by our cloud platform provider.

SHARED PUBLICLY

SLA / SLO

Is there a published SLO for the service?
We have external SLAs for products, not services. Each service should have an internal SLO, which should be the same or stricter than what is needed to support the external SLA. *Does the team understand how the product's SLA relates to their service?*

Is the SLO defensible?
Is it measurable? Is it compatible with downstream dependencies and external services? *Can the service achieve the SLO based on historical performance?* *What failure cases can be tolerated while still keeping the SLO?*

Observability

Metrics

Does the service export metrics?
All services should export Prometheus metrics using the official Prometheus client libraries. We should follow [Prometheus best practices in relation to metric names](#). *We want to avoid super high cardinality metrics – no user IDs in labels, please. How many series does each instance of the service export? Are there labels with unbounded cardinality?*

SHARED PUBLICLY

Do the exported metrics allow for RED Method style analysis?
We want to see error rate and latencies for all services, for incoming requests and outgoing requests (i.e. observe RED for the service's backends, if there are any). *See The RED Method for more info.*

Alerts

Do you use Prometheus alerting? Are the alerts version controlled?
All new services should use Prometheus. *Alerts should be version controlled in infrastructure configuration repo, in either YAML or Jsonnet please.*

Are alerts routed to the correct place?
Alerts should be routed by namespace, and hence alerts must retain the namespace label. Warning alerts, and all alerts for non-production systems, should be routed to your specific team's channels. Critical production alerts should go to your team's on-call rotation's pager.

Do you use SLO-based alerting?
You should. This blog post is a good starting point. Check out the linked resources.

Dashboards

Are dashboards version controlled?
As JSON or Jsonnet and version controlled in infrastructure configuration repository, or a

SHARED PUBLICLY

separate mixin vendored in.

Are there RED Method-style dashboards for the service?
Every service should have a RED Method-style dashboard featuring Request Rate, Error Rate and Latency for various request paths and microservices.

Are the dashboards available in our monitoring system?
As a minimum you dashboards should be automatically deployed. Link to your dashboards below.

Logs

Does the service emit logs?
Your service should emit logs to stderr / stdout, so they are automatically picked up by Loki. *You should emit logs in logfmt or JSON format, please. Grafana and Loki understand these formats. If you're using Golang, we encourage gokit logging (meaning you should use it for new projects, but there is no strict requirement to migrate existing projects as long as they meet the other requirements here).*

Do your jobs emit credentials or secrets in their logs?
Logs shouldn't contain sensitive information, so we must take care to not emit things like https usernames and passwords etc to our logs.

SHARED PUBLICLY

External Services

Are the external dependencies sufficiently monitored and is alerting set up?
eg using stackdriver_exporter to monitor Bigtable, or monitoring CloudSQL instances using mysqld_exporter etc

On-call & Incident Response

Do you have follow-the-sun on-call shifts setup?
Every service needs 24x7 on-call coverage. We recommend 2x12 hours shifts. These are usually provided by the team you are a member of.

Is the on-call rotation adequately staffed?
We recommend people are on-call no more frequently than once every 3 weeks, requiring a minimum of 6 people (3 per follow-the-sun location) for an on-call shift. If you can't meet this, you should look for another team to pair up with. *We recommend a maximum of 16 people (8 per location) on-call rotation; being on-call less frequently than once every 8 weeks is undesirable. If you grow beyond this size, you should think about splitting up your on-call shifts.*

Are you using PagerDuty/OpsGenie/VictorOps/...?
Is there an escalation path for the on-call rotation? We recommend unacknowledged pages to propagate to managers. *Is the on-call config version controlled? It should be configured with terraform in infrastructure*

SHARED PUBLICLY

configuration repo. The schedule itself (members of rotation, overrides, …) is usually configured manually.

Have the individuals on-call received adequate training on how to handle the specific alerts?
Runbooks are a good start here. Normally a generic engineer training is not sufficient.

Does the service have an entry on the Cloud status page?
This only applies if directly or indirectly user-facing. *NB each service should be represented here, but not necessarily with its own entry; some entries will cover multiple services.*

Is there an escalations channel for the customer support enquiries?
Generally this will run along team boundaries, so there should already be one.

If the service has been in production for more than one month:

Have recent outages been followed up with a post mortem?
We practice blame-free post mortems at least when we experience user-visible outages. *Were action items from the post mortem followed up on? Link any post mortems here.*

SHARED PUBLICLY

Over the past month, has the service generated less than 2 pages per day on average?
We require that on-call load is less than two incidents per shift. NB your service is unlikely to be the only service handled by your team.

State Management

If you're service uses a databases eg GCS, RDS etc

Are there (sufficiently frequent) backups? Have you tested restore?
Backups should be placed in an object store (e.g. GCS, S3, ...).

Should the data in the database be exported to BigQuery/Redshift/... for BI?
You should evaluate if the state you store is useful for analysing user behaviour, so we can improve the product.

Feedback

Did you find this checklist useful?
The aim of this checklist is to help you catch unknown unknowns; this isn't an arbitrary box-checking exercise. Did you learn anything useful?

What do you think is missing from this checklist?

SHARED PUBLICLY

Is there anything you were expecting to see here that we missed? This is only the first iteration of the PRR process, we really want your feedback.