Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Adding Observability to the Managed Tokens Service

Shreyas Bhat

Scientific Computing Monitoring Workshop

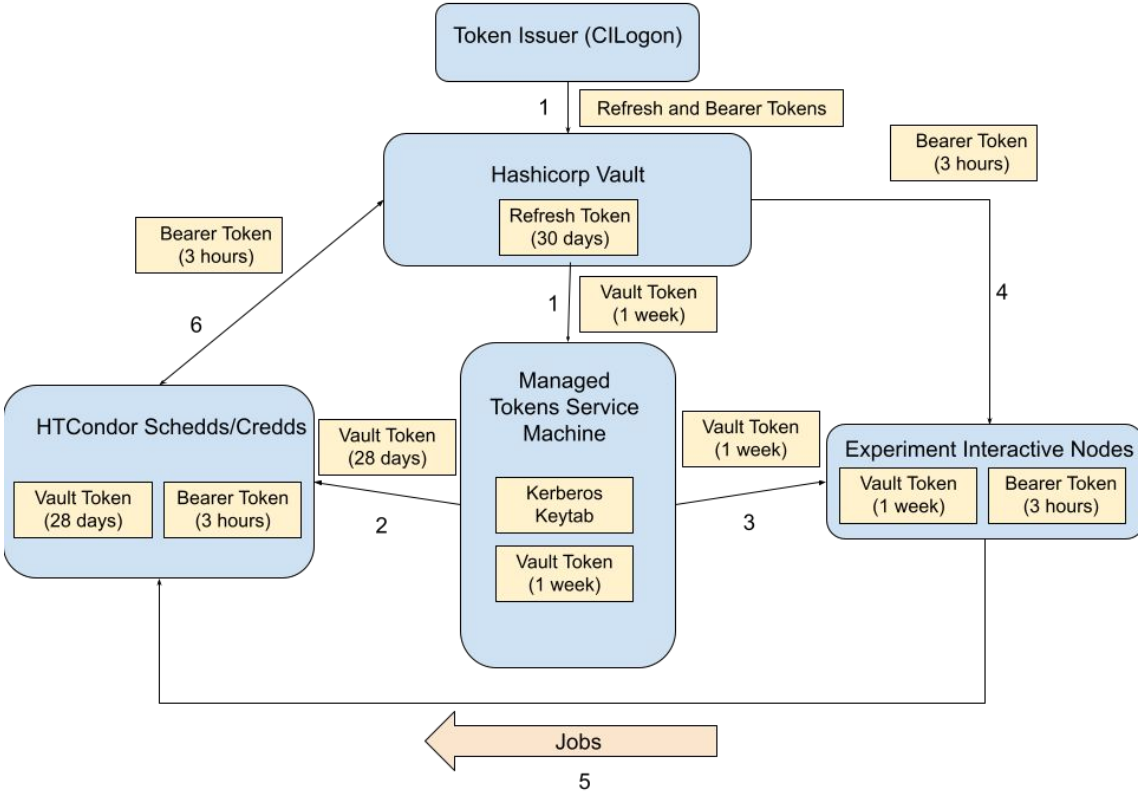
July 24, 2024

FABRIC FOR FRONTIER EXPERIMENTS

What is the Managed Tokens Service?

- Store production vault tokens in condor *credds*
- Push production vault tokens to interactive nodes, keep them refreshed
- Written in Go (performant, easy/safe concurrency, avoid Python “dependency hell”)

Where the service fits in



Executables of Managed Tokens Service

- **`token-push`:**
  - Stores vault token in HTCondor credd for experiment/role combination and push copies of vault token to interactive nodes
  - Notifies experiment stakeholders and operators if there is any error
  - Runs every hour.
- **`refresh-uids-from-ferry`:**
  - Queries FERRY to pull down the applicable UIDs for the configured UNIX accounts
  - Runs daily each morning
- **`run-onboarding-managed-tokens`:**
  - Onboards a new experiment or experiment account to the Managed Tokens Service
  - Run as needed

Normal operation of *token-push*

1. Uses a kerberos service principal to authenticate to the Hashicorp vault server
2. Sets appropriate credkey in `HTGETTOKENOPTS` based on kerberos principal → Lets `condor_vault_storer/htgettoken` know where the refresh token is in the vault
3. `condor_vault_storer`
    a. Resets the refresh token in the vault and obtains a vault token
    b. Stores this vault token in the HTCondor *credd*
4. Pushes this vault token to experiment interactive nodes
5. Notifies experiment stakeholders and/or operators if there is any error in any of the previous steps

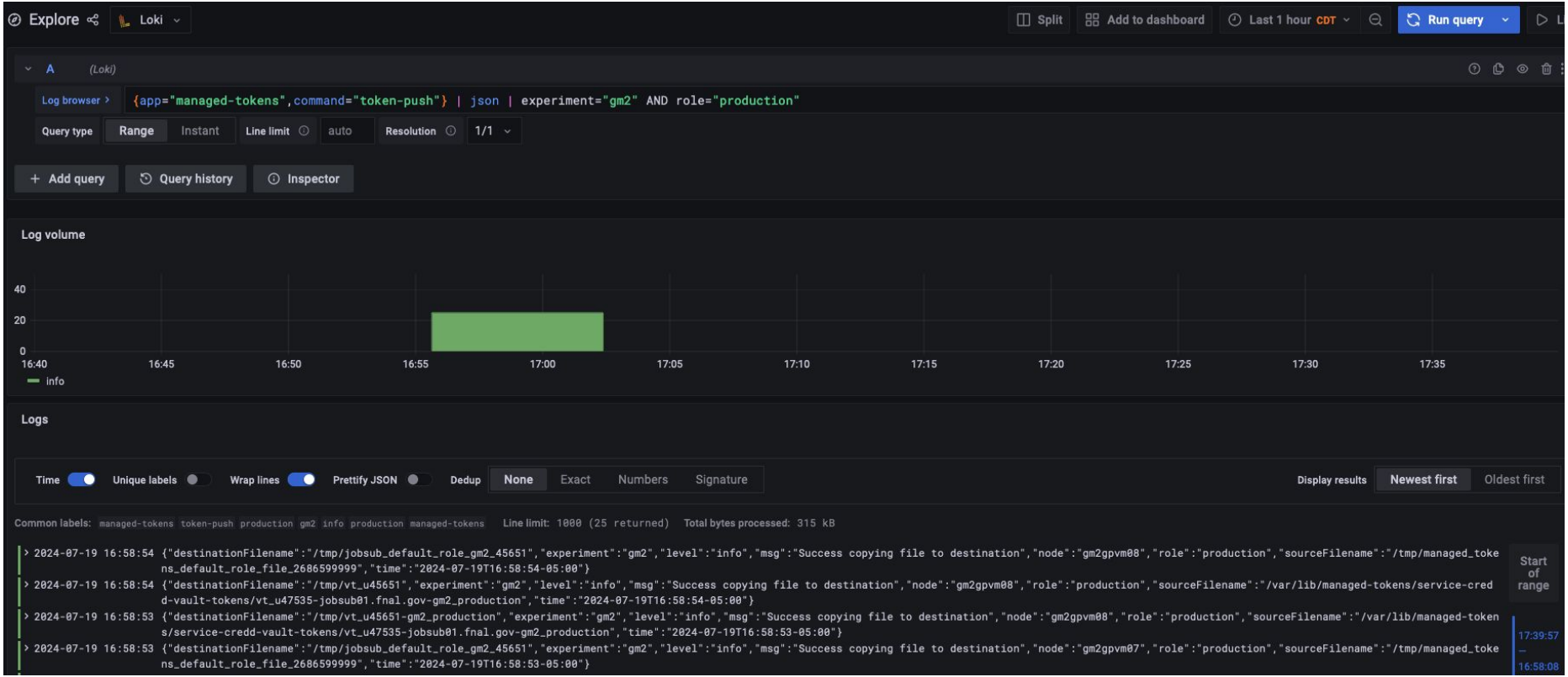
Components to keep track of

- All of the operations on the last slide plus:
- Configuration parsing
- Local sqllite database reads/writes
- Notification generation/sending
- Most of these operations (with exception of `condor_vault_storer`) are done concurrently
- How do we monitor all of this?

Logs with Loki

- Standard logs at /var/log/managed-tokens on managed tokens machine
  - One regular log, one debug log per executable
- But searching through those can be tedious: Send logs to **Loki**
- Queryable log store FE via LogQL:
  - `{app="managed-tokens",command="token-push"} | json | experiment="gm2" AND role="production"`
- Simply required adding a configured Loki hook to existing logging library - INFO and higher-level logs are sent to Loki

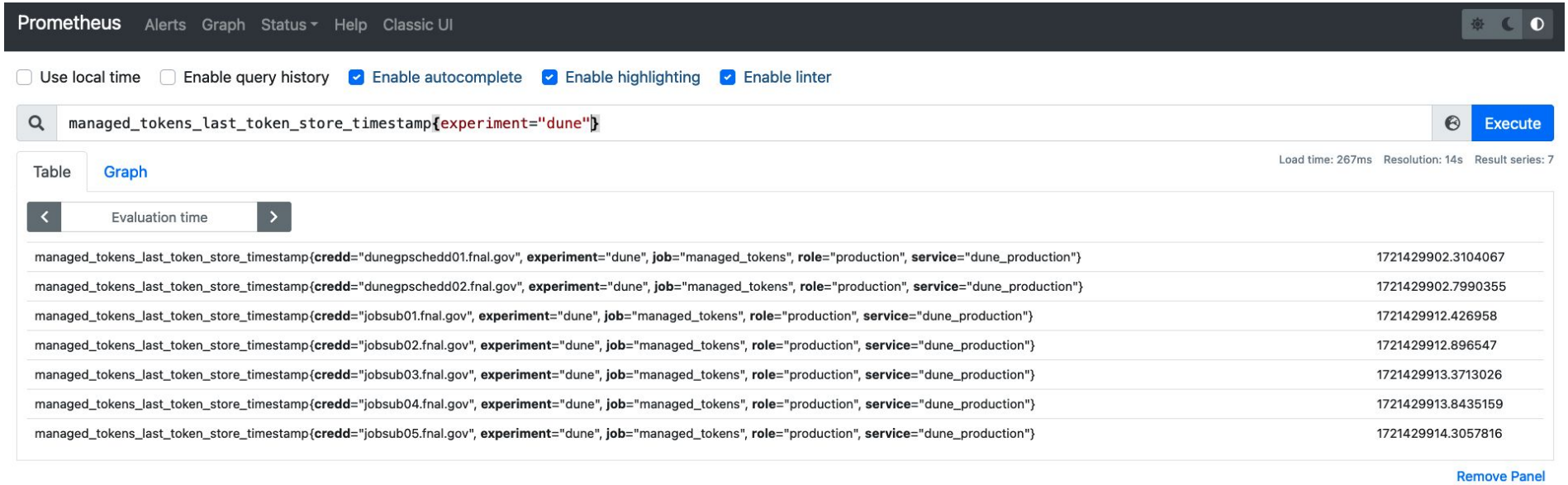
Logs with Loki (2)



Metrics

- Prometheus metrics pushed each run to a Landscape pushgateway
- Standard prometheus setup is for web services; for batch processes like this, need to use pushgateway: extra webserver that can receive metrics, then serves them to normal Prometheus server
- Metrics
  - `managed_tokens_stage_duration_seconds`: Per executable, per stage (setup, processing, cleanup). How long each stage took to run.
  - `managed_tokens_last_ferry_refresh`: Timestamp of when `refresh-uids-from-ferry` executable last got information from FERRY.
  - `managed_tokens_failed_services_push_count`: Count of how many services registered a failure to push a vault token to a node in the current run of `token-push`. Basically, a failure count.
  - `managed_tokens_last_token_push_timestamp`: Timestamp of when `token-push` last pushed a particular service vault token to a particular node.
  - `Managed_tokens_last_token_store_timestamp:` Timestamp of when `token-push` last stored a particular service vault token in a particular credd.
  - And more!

Metrics (2)

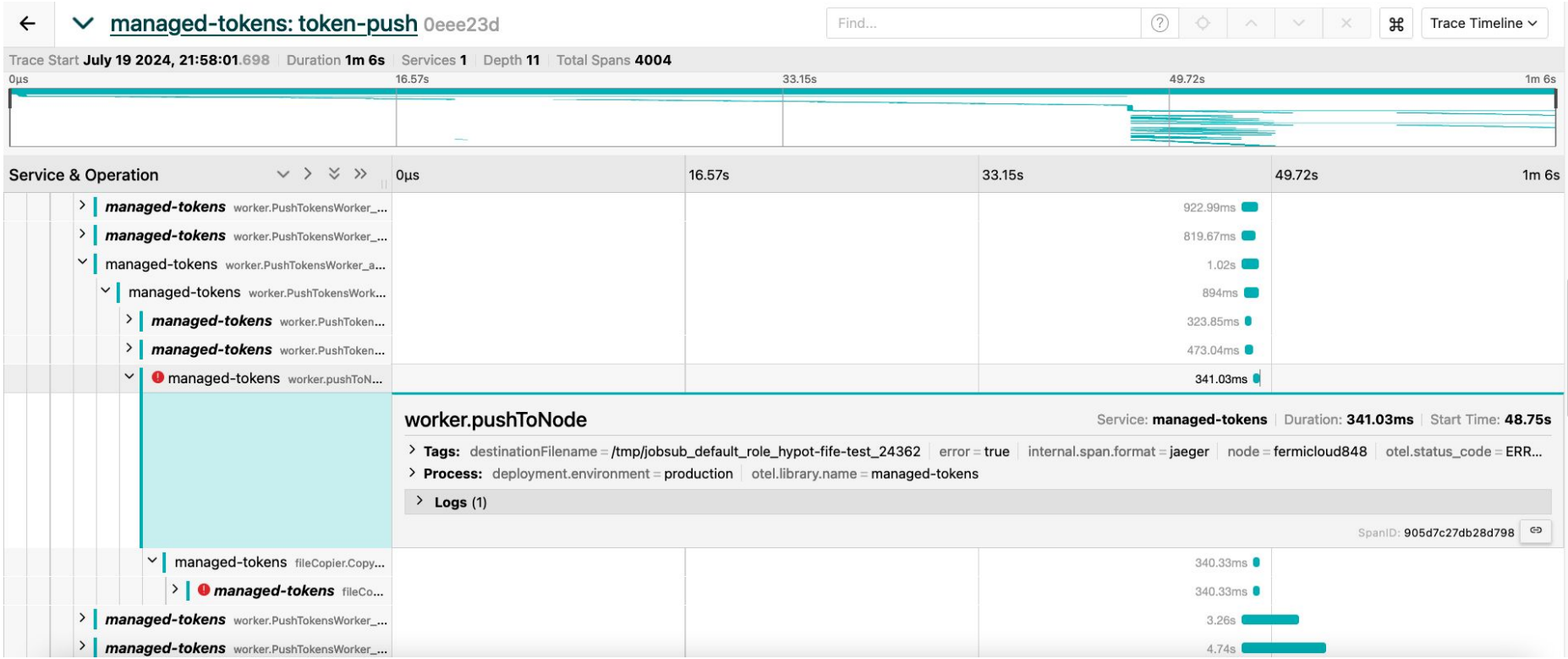

Prometheus
Alerts
Graph
Status
Help
Classic UI
Use local time
Enable query history
Enable autocomplete
Enable highlighting
Enable linter
managed_tokens_last_token_store_timestamp{experiment="dune"}
Execute
Table
Graph
Load time: 267ms
Resolution: 14s
Result series: 7
Evaluation time
managed_tokens_last_token_store_timestamp{credd="dunegpschedd01.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429902.3104067
managed_tokens_last_token_store_timestamp{credd="dunegpschedd02.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429902.7990355
managed_tokens_last_token_store_timestamp{credd="jobsub01.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429912.426958
managed_tokens_last_token_store_timestamp{credd="jobsub02.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429912.896547
managed_tokens_last_token_store_timestamp{credd="jobsub03.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429913.3713026
managed_tokens_last_token_store_timestamp{credd="jobsub04.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429913.8435159
managed_tokens_last_token_store_timestamp{credd="jobsub05.fnal.gov", experiment="dune", job="managed_tokens", role="production", service="dune_production"} 1721429914.3057816
Remove Panel

Tracing

- While metrics and logs give great insight into issues that crop up, traces help visualize WHERE in the program flow something went wrong
- Serve as a visual index rather than grepping through the logs
- Then, for more details, you can go to the logs if needed

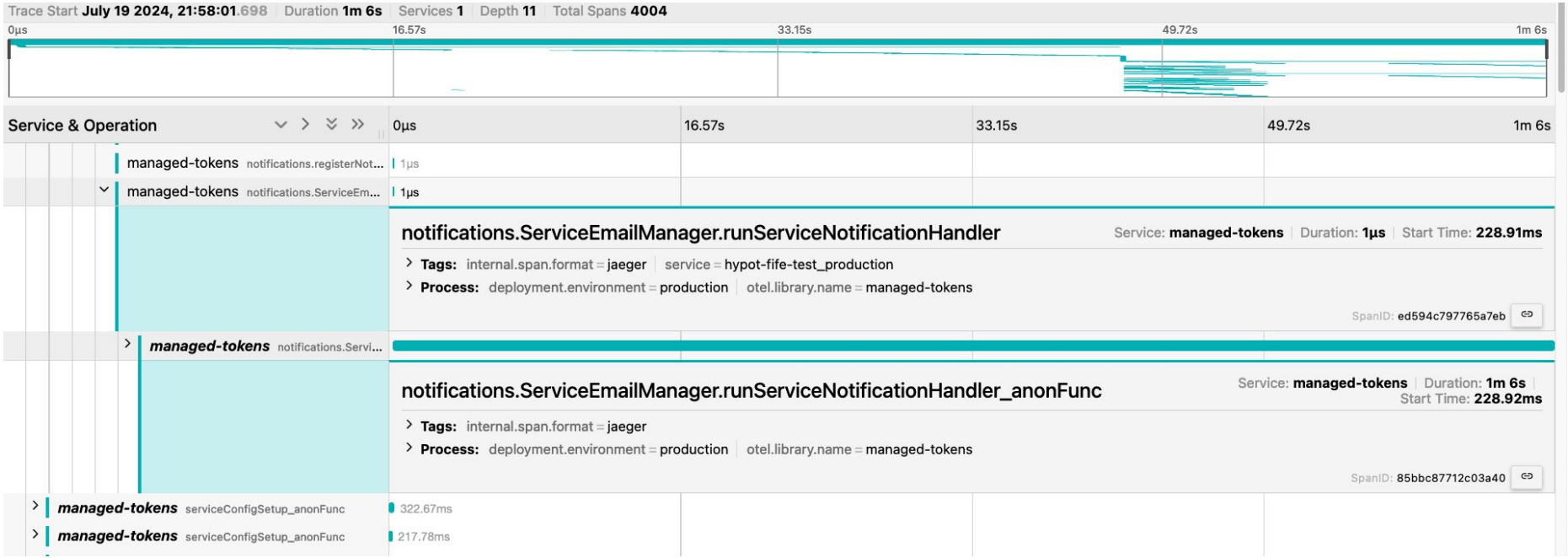
- We added OpenTelemetry tracing that gets sent to Landscape Jaeger Tracing collector
- For example: https://landscape.fnal.gov/jaeger/trace/0eee23d1cfd43126e6dc3d2d184f9007

Tracing (2)



Fermilab

Tracing to visualize concurrency (3)



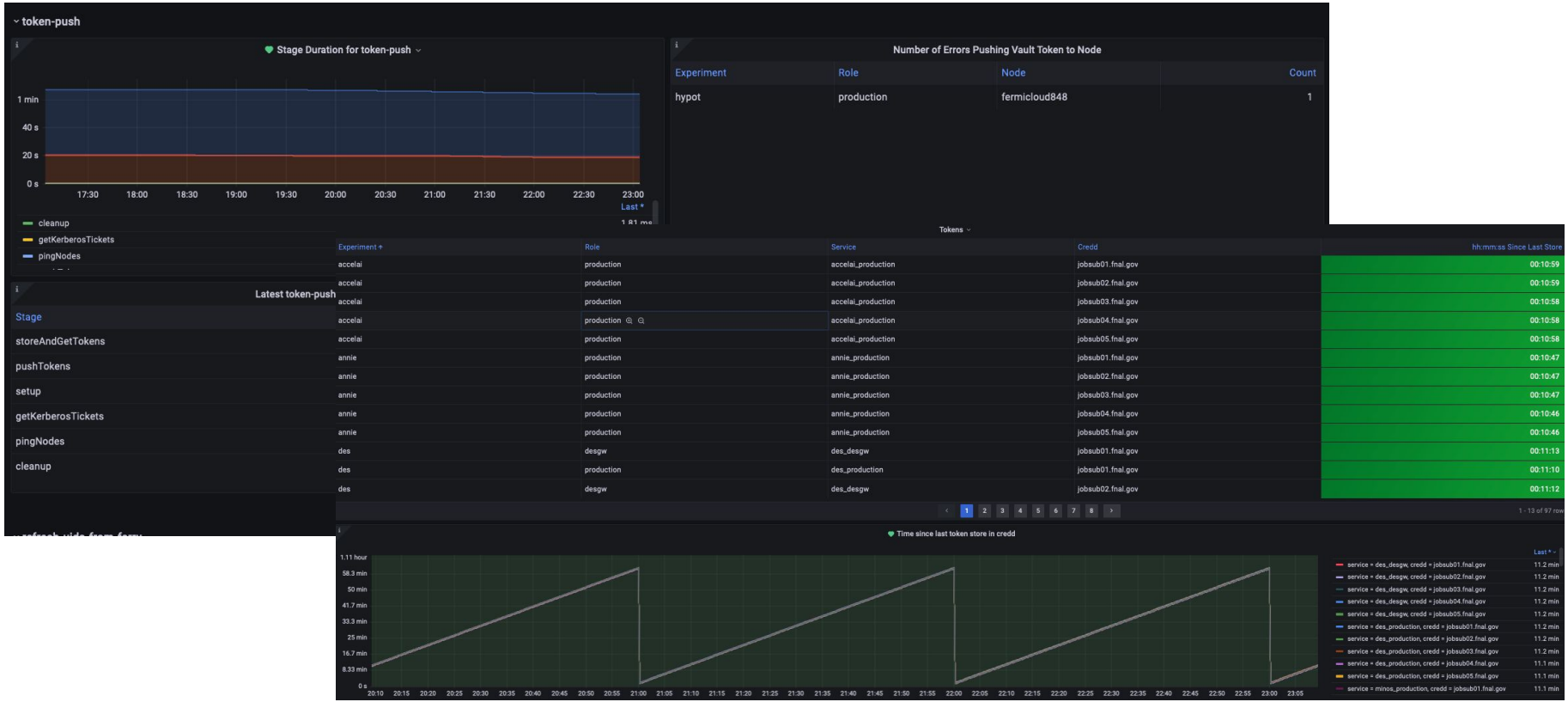
We can see here that a *hypot-fife-test_production* `ServiceNotificationHandler` is launched concurrently and stays alive for almost the entire run - waiting for messages, like the one the previous slide's error would have generated - you can SEE the concurrency!

Fermilab

Dashboards

- Fifemon dashboards:
  - Last token push by service/node: https://fifemon.fnal.gov/monitor/d/QOV9N_iIz/managed-tokens-service-time-since-last-token-push?orgId=1
  - Last token store in credd by service/credd (with alerts after 3 hrs of failure): https://fifemon.fnal.gov/monitor/d/w4mAD_iIk/managed-tokens-service-time-since-credd-store
  - Service Health Dashboard: https://fifemon.fnal.gov/monitor/d/bGDwH9mSz/managed-tokens-service-service-health

Dashboards (2)



Fermilab

Alerting

- Use dashboards to send alerts to #fifealerts Slack channel
  - Time since a token was stored in condor *credd* > 3 hrs
  - Any stage from *refresh-uids-from-ferry* takes > 5 min
  - Time since UIDs refreshed from FERRY > 2 days
- Also, original notifications from *token-push* itself:
  - Emails to stakeholders, FIFE
  - Abridged version sent to #fifealerts

Future

- Pretty much done - no major work left in adding observability
- TODO: Export traces to new Landscape OpenTelemetry collector rather than current Landscape Jaeger collector
  - One function change, since library APIs between Jaeger/OpenTelemetry are nearly-identical
  - Then rest of tracing code in *managed tokens* should seamlessly plug-in

Summary

- Managed tokens service executables are multi-threaded with numerous concurrent operations running at any time
- Adding observability has helped us pinpoint issues very quickly when they occur:
  - *Alerting/dashboards* fed by *metrics* tells us that something went wrong
  - *Traces* show us very quickly in which components of which executable the issue occurred
  - *Logs* sent to Loki (INFO level) let us quickly find the exact operation within the relevant component that went wrong
  - Debug logs on the deployment machine can help us look at details of faulty operation
- All of these components working together has saved a LOT of troubleshooting time

Resources

- Github repo: https://github.com/fermitools/managed-tokens
- Wiki page: https://fifewiki.fnal.gov/wiki/Managed_Tokens_Service
- Loki: https://grafana.com/docs/loki/latest/
- Prometheus: https://prometheus.io/
- Prometheus Pushgateway: https://prometheus.io/docs/practices/pushing/
- Using Metrics in Grafana: https://prometheus.io/docs/visualization/grafana/
- OpenTelemetry Tracing: https://opentelemetry.io/docs/

Thank you!

Extra Slides

Loki Configuration with logrus

```
244             // Loki.  Example here taken from README: https://github.com/YuKitsune/lokirus/blob/main/README.md
245             lokiOpts := lokirus.NewLokiHookOptions().
246                     // Grafana doesn't have a "panic" level, but it does have a "critical" level
247                     // https://grafana.com/docs/grafana/latest/explore/logs-integration/
248                     WithLevelMap(lokirus.LevelMap{log.PanicLevel: "critical"}).
249                     WithFormatter(&log.JSONFormatter{}).
250                     WithStaticLabels(lokirus.Labels{
251                             "app":         "managed-tokens",
252                             "command":     currentExecutable,
253                             "environment": devEnvironmentLabel,
254                     })
255             lokiHook := lokirus.NewLokiHookWithOpts(
256                     viper.GetString("loki.host"),
257                     lokiOpts,
258                     log.InfoLevel,
259                     log.WarnLevel,
260                     log.ErrorLevel,
261                     log.FatalLevel)
262
263             log.AddHook(lokiHook)
```

Code to add loki hook to logrus logging library

[Link to code on Github]

Configuration for loki in managed tokens config file

```
loki:
  host: "http://fifelog.fnal.gov:3100"
```

Prometheus configuration

- Configuration to push metrics to pushgateway at http://fifelog.fnal.gov:9091
- Set up `MetricsRegistry` using prometheus Go client library and prometheus Go pushgateway library
- Set up metrics in each executable, like this (link to Github)

Tracing configuration

- Configuration to export all traces to https://landscape.fnal.gov/jaeger-collector/api/traces : note - there is now an OpenTelemetry-compatible trace collector that should be used instead
- Set up `TracerProvider` to export traces (example)
- Set up executables to use this `TracerProvider` when creating new spans (see here)
- In each operation/function, start a new span (example)