
# pSConfig Overview

Andy Lake — andy@es.net

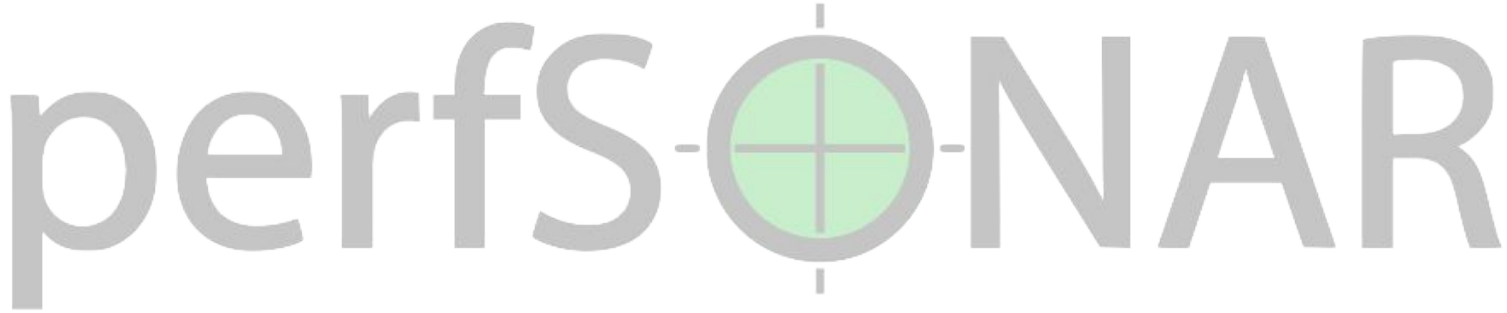
June 5th, 2019





# Outline

- pSConfig Basics
- What's New



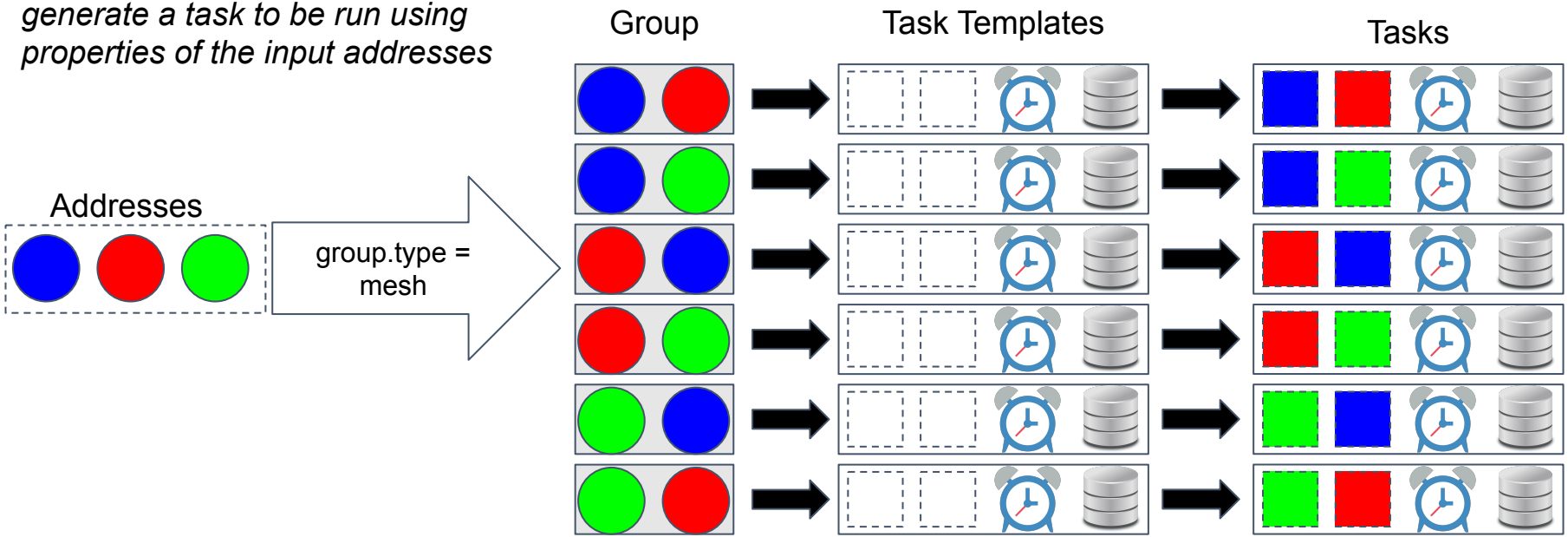

# pSConfig Basics

# What is pSConfig?

*pSConfig* is a **template** framework for describing and configuring a **topology** of **tasks**

# pSConfig: Creating Tasks

*For each pair in the group, we generate a task to be run using properties of the input addresses*



# pSConfig Real-World Example

As example let's say we have four hosts we want to run throughput and latency tasks. Some have a single interface and some have multiple, with each interface color-coded with the tests we want run below. They will publish their results in a central esmond archive.

esmond.archive.perfsonar.net

lat1.perfsonar.net

thr1.perfsonar.net

thrlat1.perfsonar.net

lat2.perfsonar.net
thr2.perfsonar.net

| Address Color Code Key | | | |
|---|---|---|---|
| Runs Throughput tests | Runs latency Tests | Runs both | Archive |

# pSConfig Real-World Example - addresses

```
"addresses": {
  "lat1": {
    "address": "lat1.perfsonar.net"
  },
  "thr1": {
    "address": "thr1.perfsonar.net"
  },
  "thrlat1": {
    "address": "thrlat1.perfsonar.net"
  },
  "lat2": {
    "address": "lat2.perfsonar.net"
  },
  "thr2": {
    "address": "thr2.perfsonar.net"
  }
}
```

# pSConfig Real-World Example - groups

```
"groups": {
  "throughput_group": {
     "type": "mesh",
     "addresses": [
        {"name": "thr1"},
        {"name": "thrlat1"},
        {"name": "thr2"}
     ]
  },
```

```
  "latency_group": {
     "type": "mesh",
     "addresses": [
        {"name": "lat1"},
        {"name": "thrlat1"},
        {"name": "lat2"}
     ]
  }
}
```

# pSConfig Real-World Example - tests

```
"tests": {
  "throughput_test": {
    "type": "throughput",
    "spec": {
      "source": "{% address[0] %}",
      "dest": "{% address[1] %}",
      "duration": "PT30S"
    }
  },
```

```
  "latency_test": {
    "type": "latencybg",
    "spec": {
      "source": "{% address[0] %}",
      "dest": "{% address[1] %}",
      "packet-interval": 0.1,
      "packet-count": 600
    }
  }
}
```

# pSConfig Real-World Example - archives

```
“archives”: {
  “esmond_archive”: {
      “archiver”: “esmond”,
      “data”: {
        “url”:
“https://esmond.archive.perfsonar.net/esmond/perfsonar/archive”
      }
   }
}
```

# pSConfig Real-World Example - schedules

```
"schedules": {
  "every_4_hours": {
      "repeat": "PT4H",
      "slip": "PT4H",
      "sliprand": true
   }
}
```

# pSConfig Real-World Example - tasks

```
"tasks": {
  "throughput_task": {
     "group": "throughput_group",
     "test": "throughput_test",
     "archives": ["esmond_archive"],
     "schedule": "every_4_hours"
  },
```

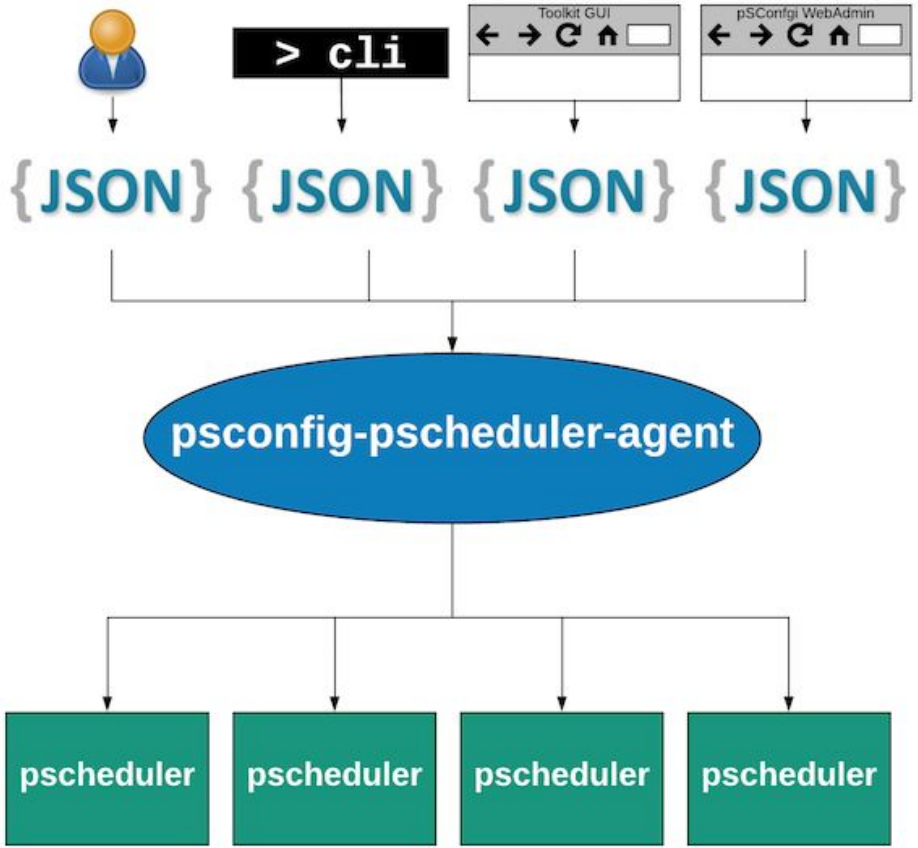
```
  "latency_task": {
     "group": "latency_group",
     "test": "latency_test",
     "archives": ["esmond_archive"]
  }
}
```

# Agents

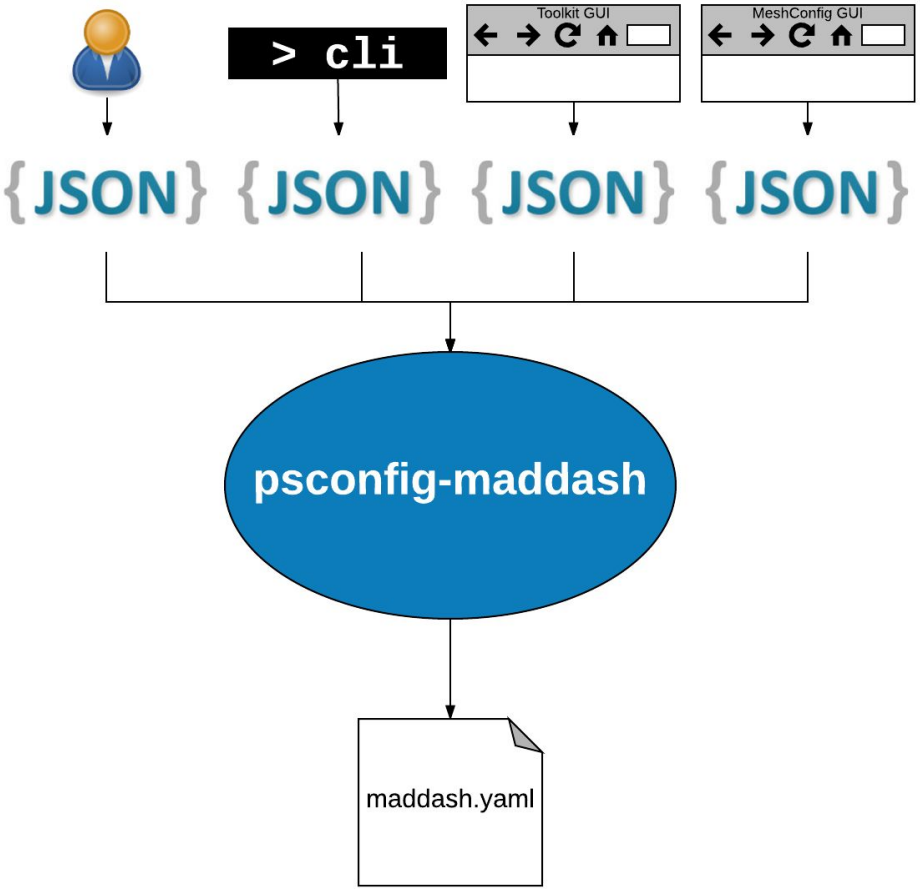
*An **agent** is software that reads one or more pSConfig templates and uses the information to perform a specific function.*

- We currently have two agents:
  - **pscheduler-agent**: It reads the template file(s) and generates pScheduler tasks
  - **maddash-agent:** It reads the the template file(s) and generates a maddash.yaml file

# pSConfig
# pScheduler
# Agent



# pSConfig MaDDash Agent



# Configuring Templates

*In order for the agent to create tasks, it must first be configured to read one or more templates. There are multiple ways to add a template depending on its location relative to the host system of the agent. These include:*

1. Configuring ***remote*** templates by supplying a URL and desired options to the agent. This is most commonly done using the `psconfig remote` command.
2. Configuring ***local*** templates that live on the agent's filesystem either using the `psconfig remote` command OR by copying the template files to a dedicated directory whose contents are automatically read by the agent.

# Remote Templates

*Adding a remote template:*

```
# psconfig remote add "https://10.0.0.1/example.json"
```

*Adding a remote template and using the archives it defines:*

```
# psconfig remote add --configure-archives "https://10.0.0.1/example.json"
```

*Listing templates:*

```
# psconfig remote list
```

*Full list of command options:*

```
# psconfig remote --help
```

# Local Templates

*Adding a local file with `psconfig remote`:*

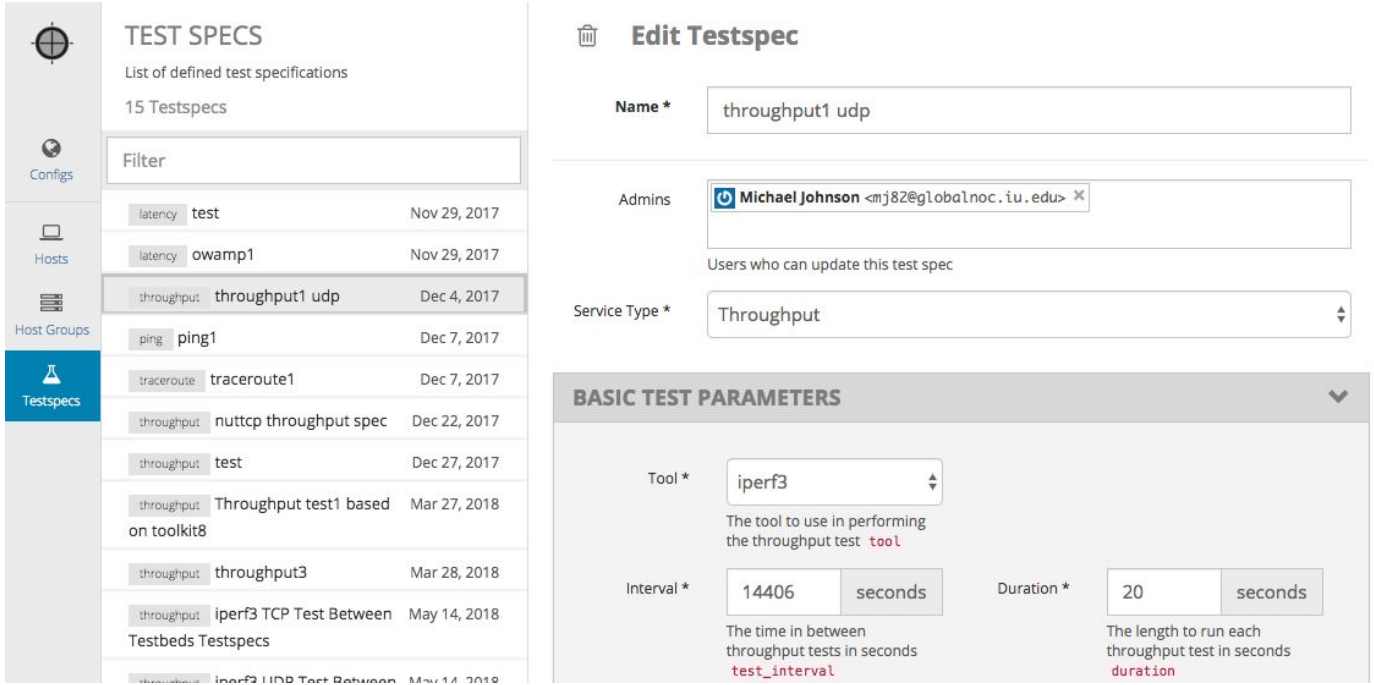
```
# psconfig remote add /path/to/example.json
```

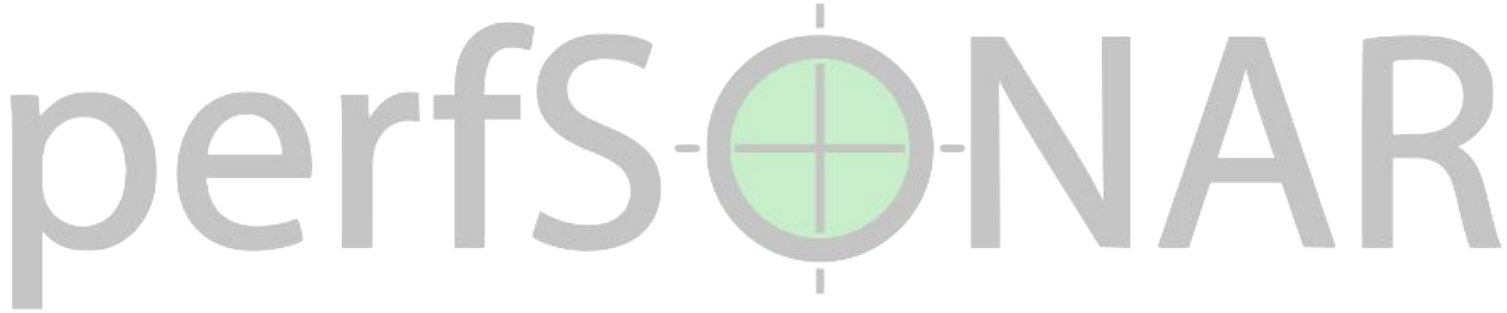
*Copying to agent include directory:*

```
# cp /path/to/example.json /etc/perfsonar/psconfig/pscheduler.d/example.json
```

```
# cp /path/to/example.json /etc/perfsonar/psconfig/maddash.d/example.json
```

# pSConfig WebAdmin (PWA)






# What's New

# Command-Line Utilities: validate

```
$ psconfig validate https://ps-west.es.net/psconfig/esnet-psconfig.json
Loading template ..... OK
Validating JSON schema ..... OK
Verifying object references ..... OK
pScheduler Validation (Quick) ..... OK

pSConfig JSON is valid
```

# Command-Line Utilities: agentctl

**$ psconfig agentctl pscheduler match-addresses ?**

match-addresses

Type: Hostname or IP Address list

Description:
List of IP addresses and/or hostnames to use when determining which tests this agent should configure. Default is all the addresses found on interfaces on this host.

Example: 10.1.1.1

# Command-Line Utilities: agentctl

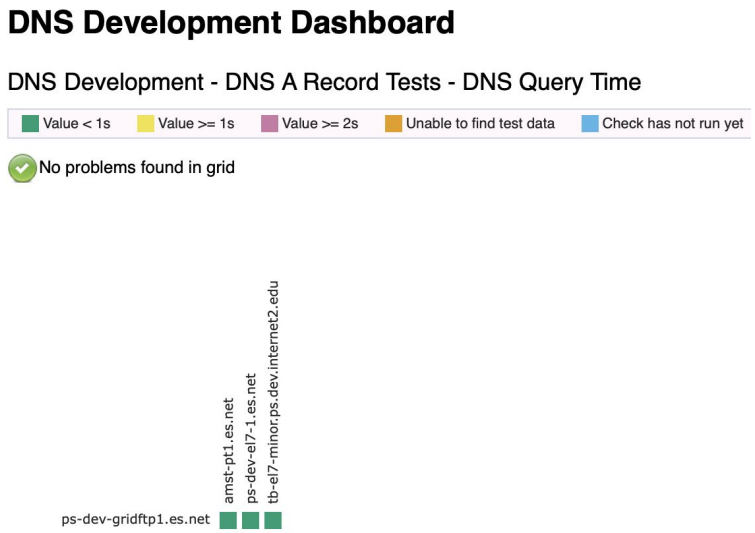
```
$ psconfig agentctl pscheduler match-addresses 10.0.0.1
Successfully set match-addresses in /etc/perfsonar/psconfig/pscheduler-agent.json

$ psconfig agentctl pscheduler --unset match-addresses
Successfully unset match-addresses in
/etc/perfsonar/psconfig/pscheduler-agent.json

$ psconfig agentctl pscheduler ?
archive-directory
cache-directory
cache-expires
…
```

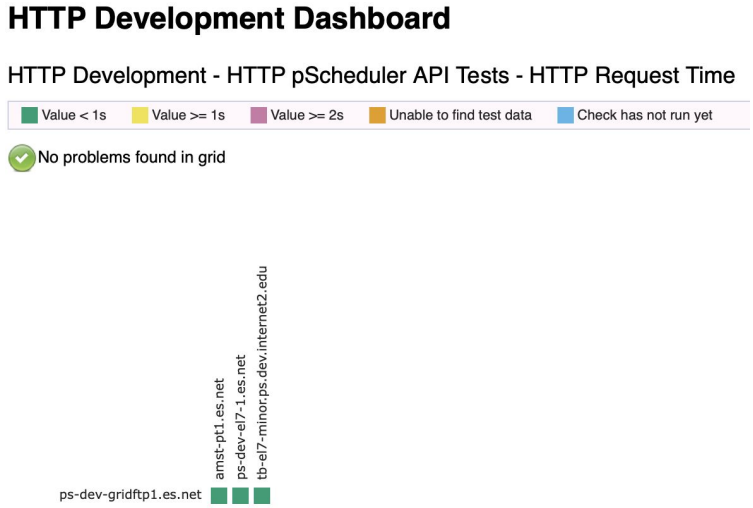
# DNS Test Example

```
"tests": {
  "dns_a_record_test": {
    "type": "dns",
    "spec": {
      "record": "a",
      "query": "{% address[1] %}"
    }
  }
}
```



# HTTP Test Example

```
"tests": {
 "http_pscheduler_api_test": {
  "type": "http",
  "spec": {
   "url": "https://{% address[1] %}/pscheduler",
   "parse": "API server"
  }
 }
}
```

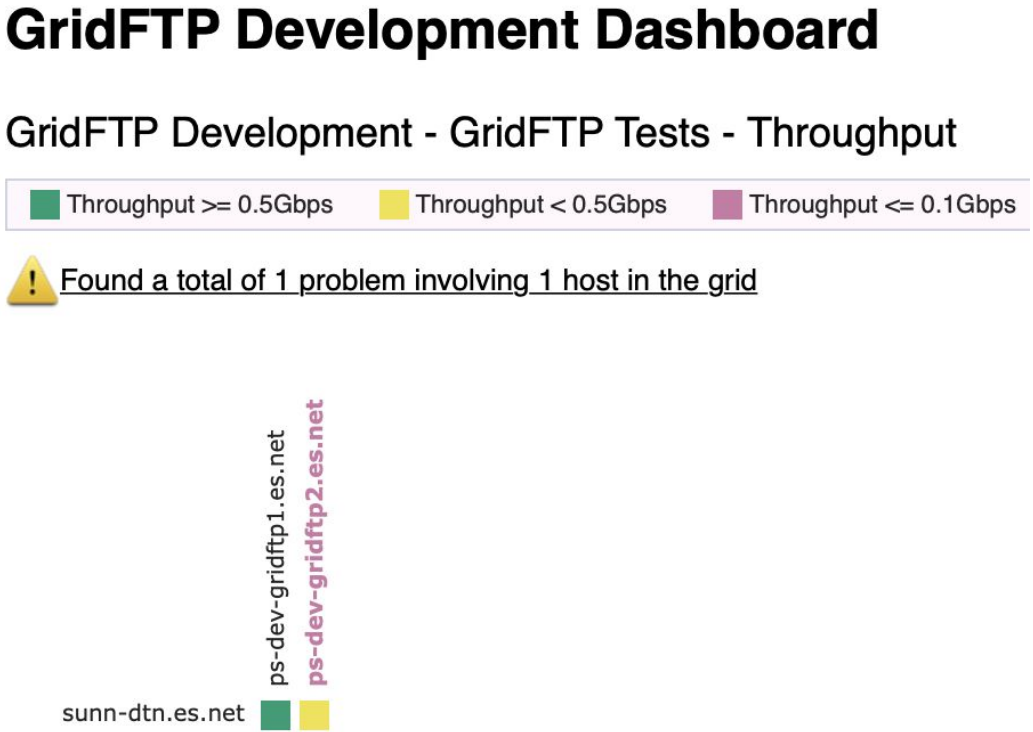


# GridFTP Test Example

```
"addresses": {
....
  "sunn-dtn.es.net": {
    "address": "sunn-dtn.es.net",
    "no-agent": true,
    "_meta": {
       "gridftp_port": 2811,
       "gridftp_filename": "/data1/1G.dat"
    }
  }
}
```

```
"tests": {
  "disktodisk_test": {
    "type": "disk-to-disk",
    "spec": {
    "source": "ftp://{% address[0] %}:{% jq
.addresses[0]._meta.gridftp_port %}{% jq
.addresses[0]._meta.gridftp_filename %}",
    "dest": "file:///dev/null",
    "timeout": "PT30S"
    }
  }
}
```

# GridFTP Test Example



# Closing Remarks

- pSConfig introduces a lot of flexibility into the types of tests we can push to hosts
- We are starting to see the benefits of that flexibility in the ability to define new types of tests and dashboards
- That flexibility introduces complexity, we are trying to provide tools to make that easier to navigate
- More work to do!

# Resources

- pSConfig/MaDDash Cheat Sheet: https://docs.google.com/document/d/1IJfe3WDClFDvwxMKk-NlAkSz0XLm4C9mffUbEvMW87c/edit?usp=sharing
- perfSONAR docs: http://docs.perfsonar.net
- PWA Intro: https://www.youtube.com/watch?v=3mjeLQdOCJQ