Search Head Clustering

Eric Woo – Senior Engineer

Manu Jose – Senior Engineer

splunk>

Disclaimer

During the course of this presentation, we may make forward looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not, be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Agenda

- What is Search Head Clustering ?
- Business Benefits of Search Head Clustering
- Cluster Operation
- Scalability and High Availability
- Configuration Management
- Q&A

Search Head Clustering

Ability to group search heads into a cluster in order to provide Highly Available and Scalable search services

MISSION CRITICAL ENTERPRISE

Business Benefits of SHC

Horizontal Scaling

Consistent User Experience

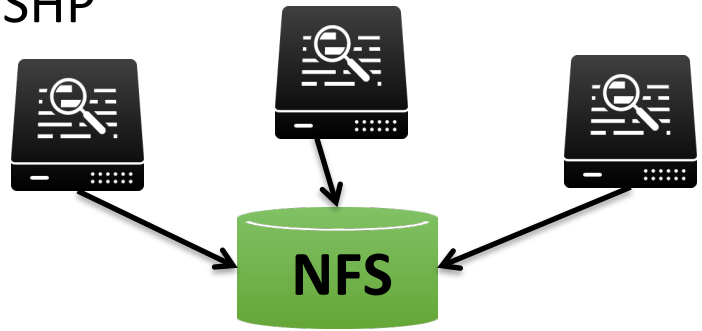
Always-on Search Services

Easy to add / manage premium contents (apps)

SHP vs SHC

Deprecated

SHP



- Available since v4.2
- Sharing configurations through NFS
- Single point of failure
- Performance issues

SHC

- No NFS
- Replication using local storage
- Commodity hardware

Design Goals

1. No Single Point of Failures
2. “One Configuration” across SH
3. Horizontal Scaling

Implementation

1. Dynamic Captain
2. Automatic Config replication across SH
3. Ability to add / remove nodes on running cluster

SHC – How does it work?

1. Group search heads into a cluster

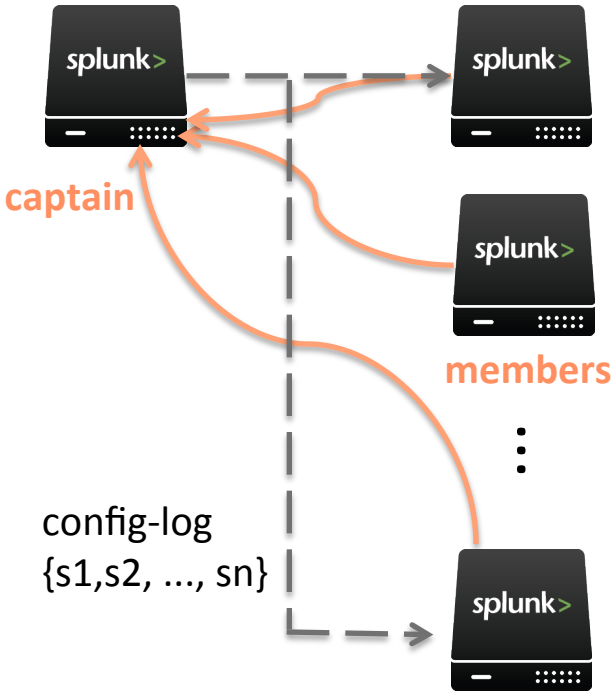
2. A captain gets elected dynamically

3. User created reports/dashboards automatically replicated to other search heads

Deploy a Cluster

splunk>

Search Head Cluster Bring up



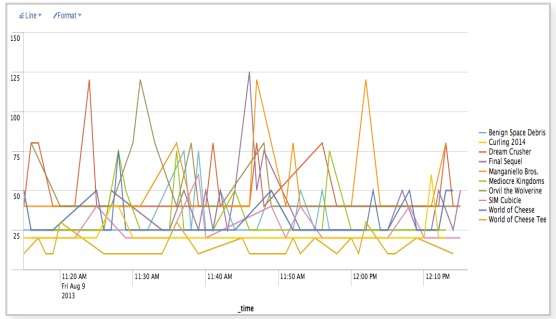
- Bootstrap captain
- Bring-up members
- Captain establishes authority
- Members join/register
- Common splunk.secret and clusterId
- CLI based cluster scale/shrink

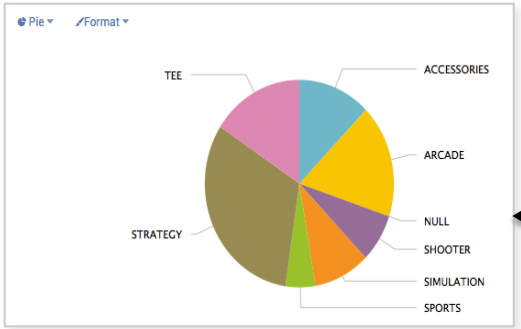
Distributed Scheduling

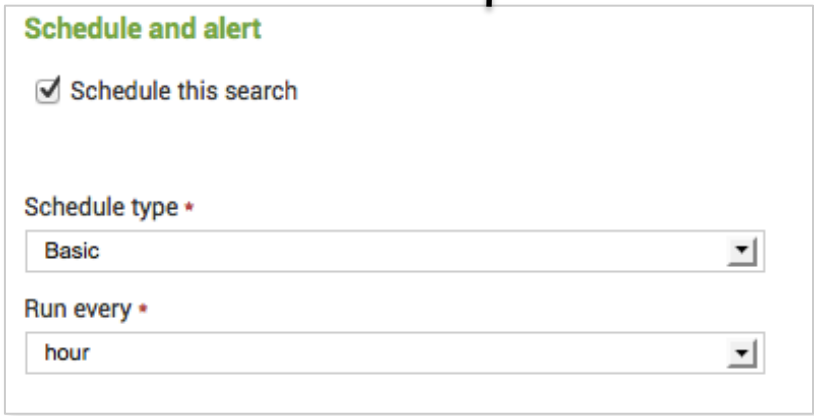
splunk>

Use Case

- Scale search capacity
- Enable more reports, dashboards, alerts
- Load balance user sessions (onboarding)

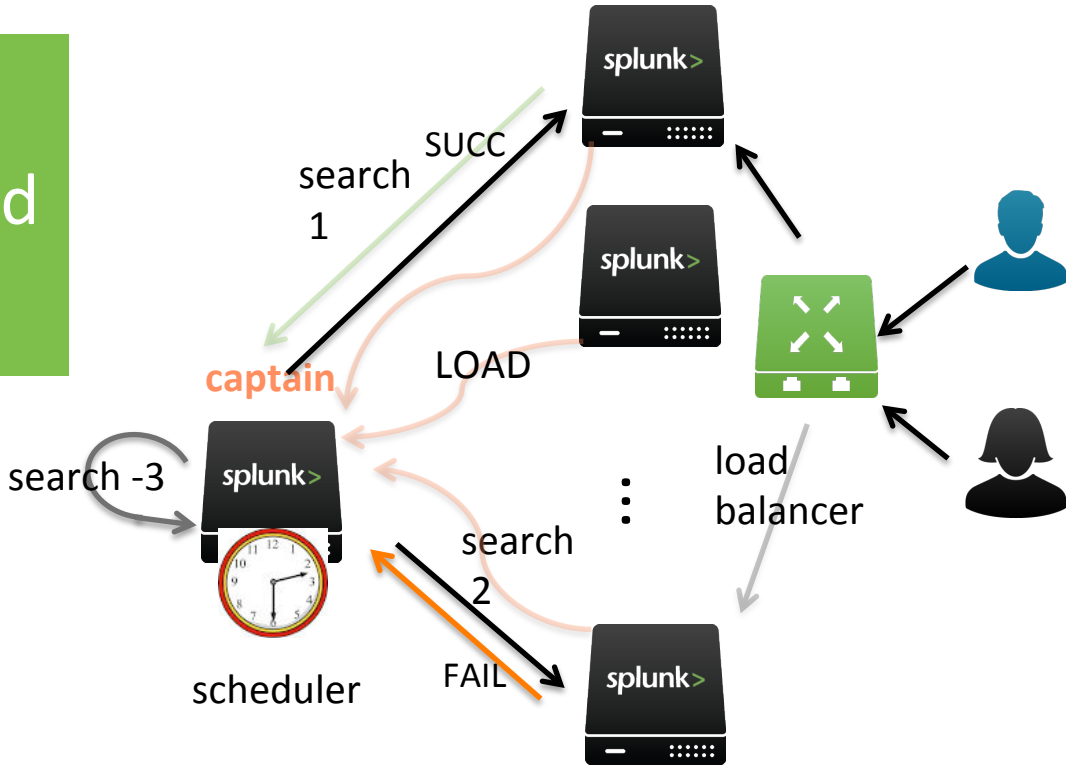






Job Scheduling Orchestration

- Captain is job scheduler
- Eliminates job-server need
- Load-based heuristic



Details

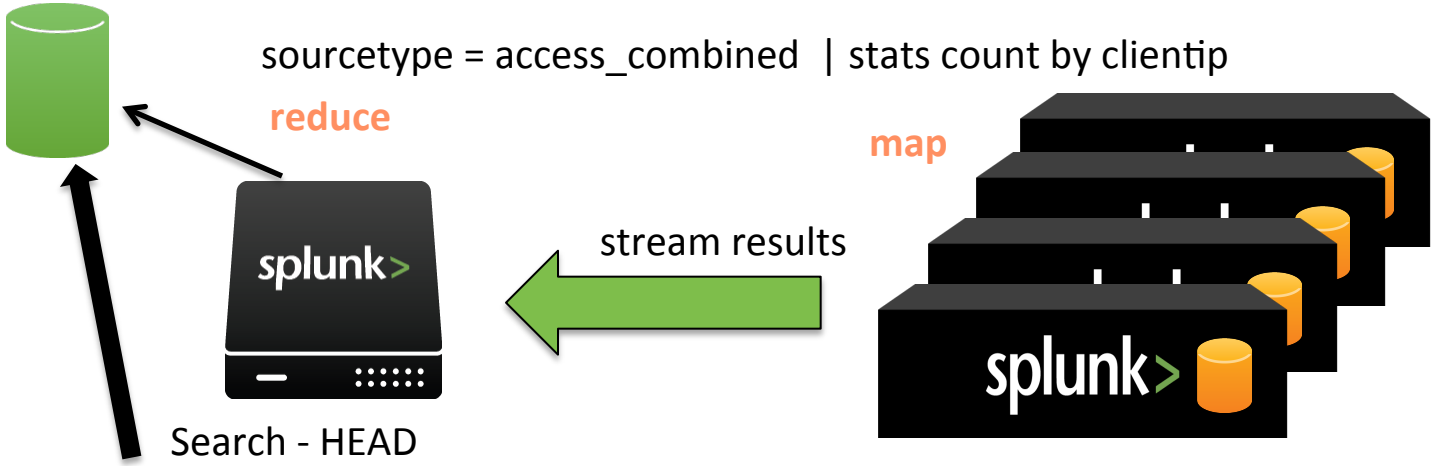
- Centralized user quota Management
- captain_is_adhoc_searchhead knob to reduce captain load
- Captain updates RA/DM summaries on indexers.
- Scheduler limits honored across the cluster
- Real time scheduled searches run one instance across cluster
- Auto-failover – New captain becomes scheduler

High Availability of Search Results

splunk>

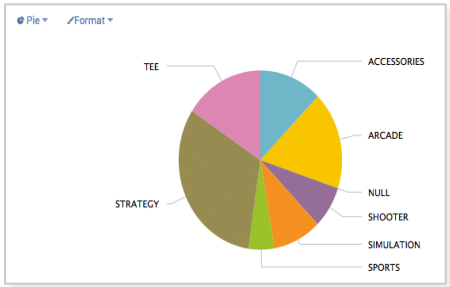
Search Results primer

$SPLUNK_HOME/var/run/splunk/dispatch/scheduler__admin__search_mysearch_at_1410708600_345



Other names:

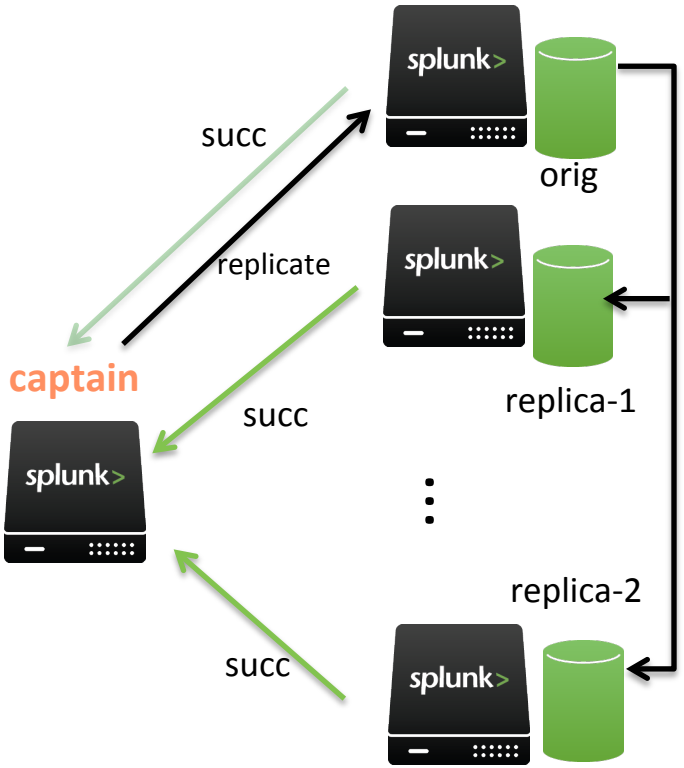
1. search results
2. search artifact
3. dispatch directory
4. SID



Dispatch dir needs to be replicated to multiple nodes to tolerate node failures

Artifact Replication

- Captain orchestrates replication
- Default replication_factor=3
- Success failure ACK'd to captain
- Asynch Replicate on Proxy
- Replication policy enforced by fixups



Centralized Cluster State

- Captain maintains a global view of alerts and suppressions and updates the list to all members
- Captain registers all the adhoc searches run in the cluster
- Captain orchestrates reaping of search artifact replicas
- GET /services/search/jobs requests on any member will proxy to captain to get complete jobs

High Availability of Cluster

splunk>

HA & Auto-Failover

Design Goals

1. No Single Point of Failure
2. Continuous Uptime
3. Consistent User Experience

Implementation

1. Dynamic Captain election
2. Auto Failover
3. Proxying for consistent view

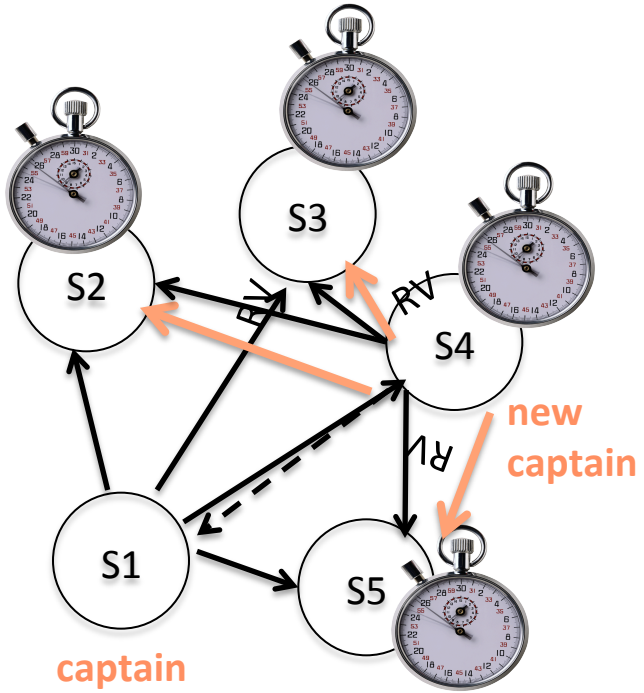
Dynamic Captain

- Raft Consensus Protocol from Stanford
  - Diego Ongaro & John Osterhout
  - Acknowledge Diego Ongaro for help!
- SHC uses RAFT for LE and Auto Failover

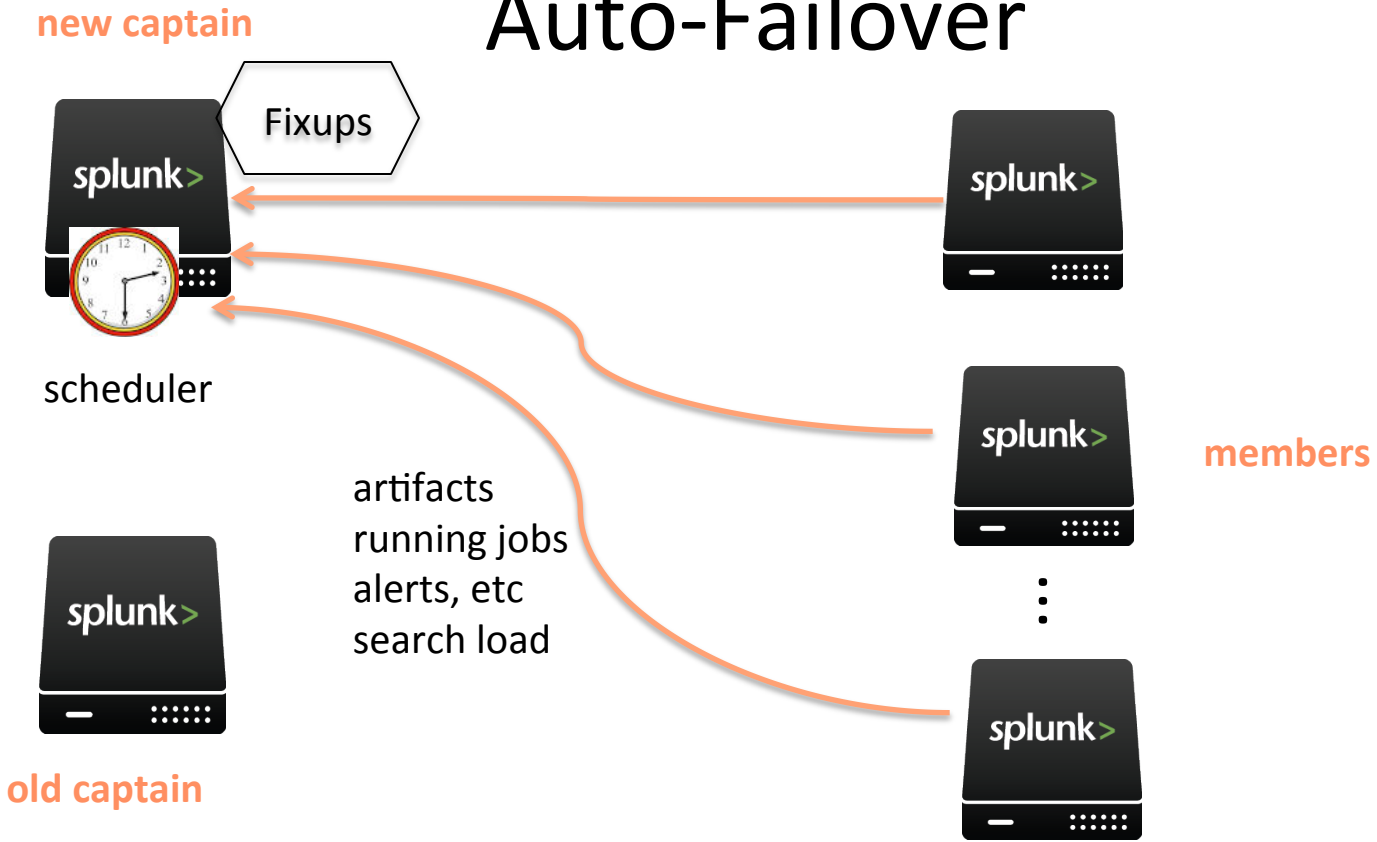
RV = Request Vote

LE = Leader Election

SHC = Search Head Clustering



Auto-Failover



Disaster Recovery – Static Captaincy

- Advised Use Cases
  - Single site cluster looses majority
  - Multi Site cluster where majority site fails
  - NOT advised for network partition due to inability to reconcile configuration changes
- Limitations
  - Manual Intervention required
  - Single Point of Failure
  - Works with or without enough members to meet replication factor

Configuration Management

splunk>

Configuration Files

- Custom user content
  - reports
  - Dashboards
- Search-time knowledge
  - field extractions
  - event types
  - Macros
- System configurations
  - inputs, forwarding, authentication

Goal

- Consistent user experience across all search heads
- Changes made on one member are reflected on all members

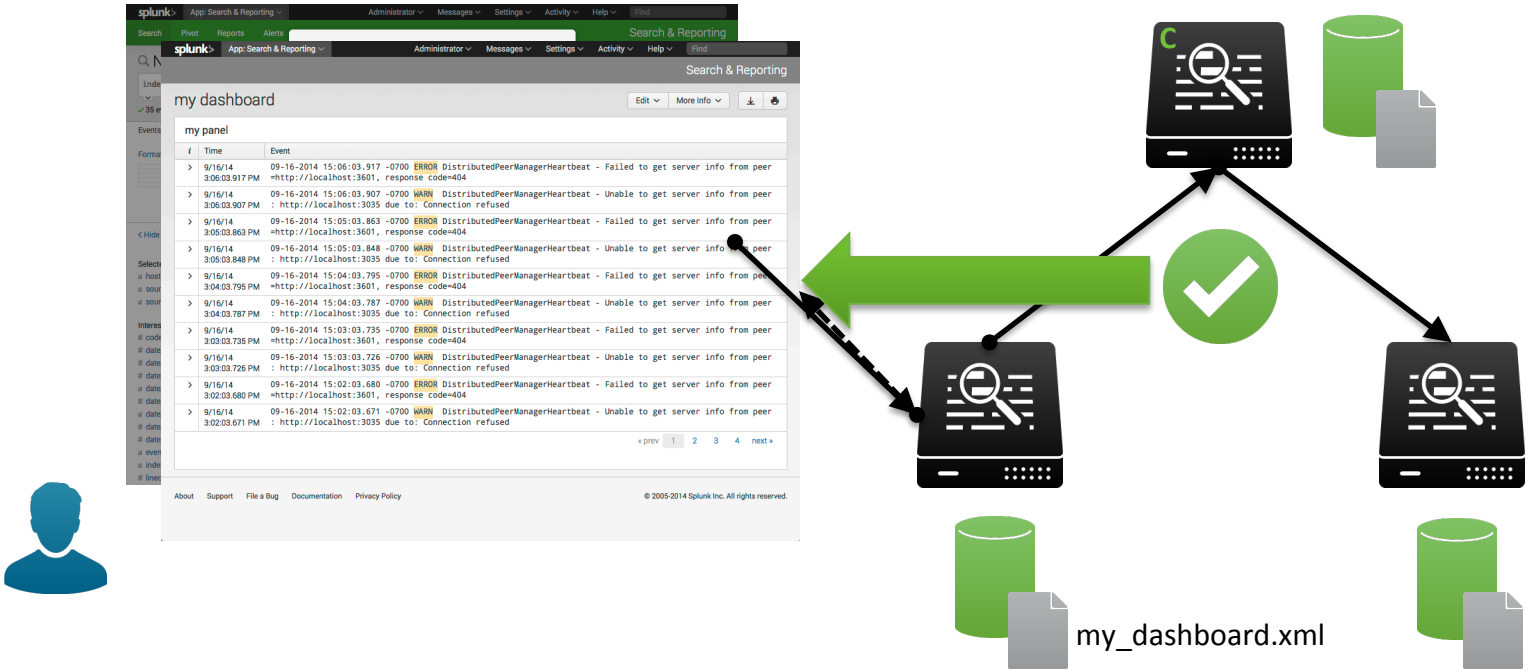
Configuration Changes

- Users customize search and UI configurations via UI/CLI/REST
  - Save report
  - Add panel to dashboards
  - Create field extraction
- Administrators modify system configurations
  - Configure forwarding
  - Deploy centralized authentication (e.G. Ldap)
  - Install entirely new app or hand-edited configuration

Search and UI Configurations

- Changes to search and UI configurations are replicated across the search head cluster automatically
- Goal: eventual consistency

Configuration Replication

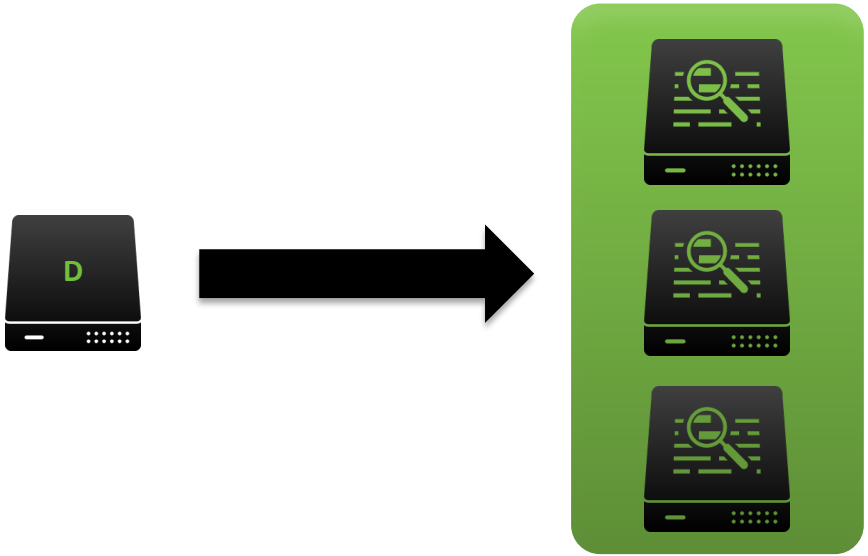


System Configurations

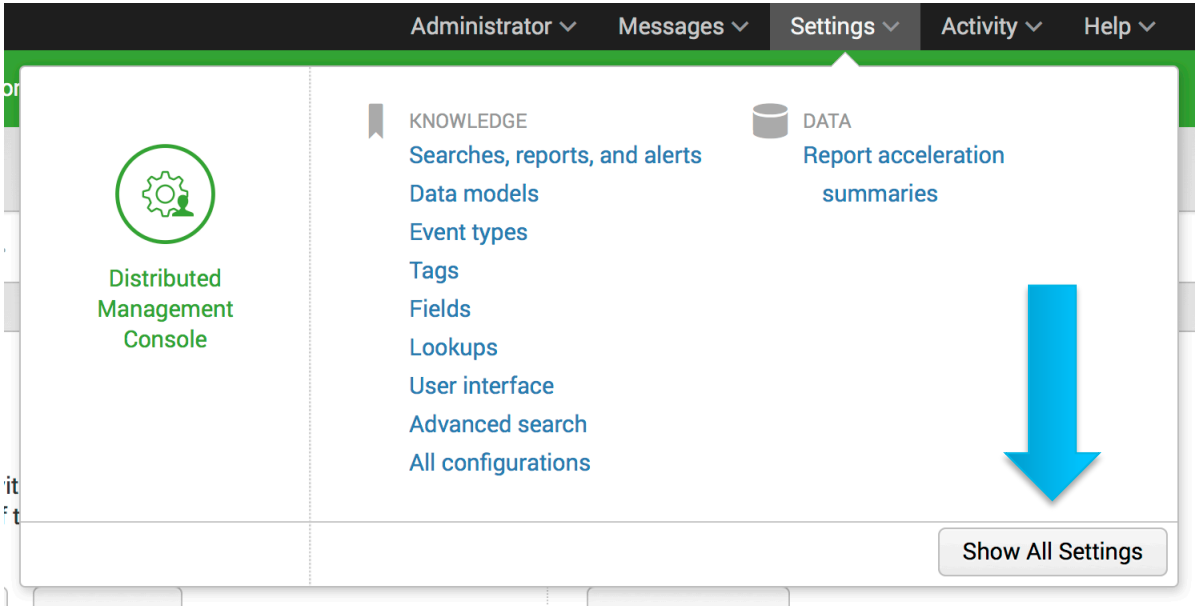
- Recall: only changes to search and UI configurations are replicated across the search head cluster automatically
- Changes to system configurations are **not** replicated automatically because of their high potential impact
- How are system configurations kept consistent, then?

Configuration Deployment

- Deployer: a single, well-controlled instance outside of the cluster
- Configurations should be tested on dev/QA instances prior to deploy



UI


Administrator
Messages
Settings
Activity
Help
Distributed Management Console
KNOWLEDGE
Searches, reports, and alerts
Data models
Event types
Tags
Fields
Lookups
User interface
Advanced search
All configurations
DATA
Report acceleration summaries
Show All Settings

Migration: User Configurations

- Task: switch from non-clustered search head to search head cluster
- Deployer: migrate user configurations to captain
- In 6.3, captain applies and replicates changes just like native changes made via UI/CLI/REST
- Migrated user configurations thus behave just like configurations created natively on the search head cluster

THANK YOU

splunk>

Stable Captaincy

- Captain Switching should be extremely rare
- Repair a problem by transfer captain without restarts!!!
  - Preference on a node to be or not to be captain
  - Node configured not to run adhoc searches as Captain hidden from load balancer
- Rolling-restart from the captain maintains the node as captain after restarts
  - Status available via CLI/audit logs

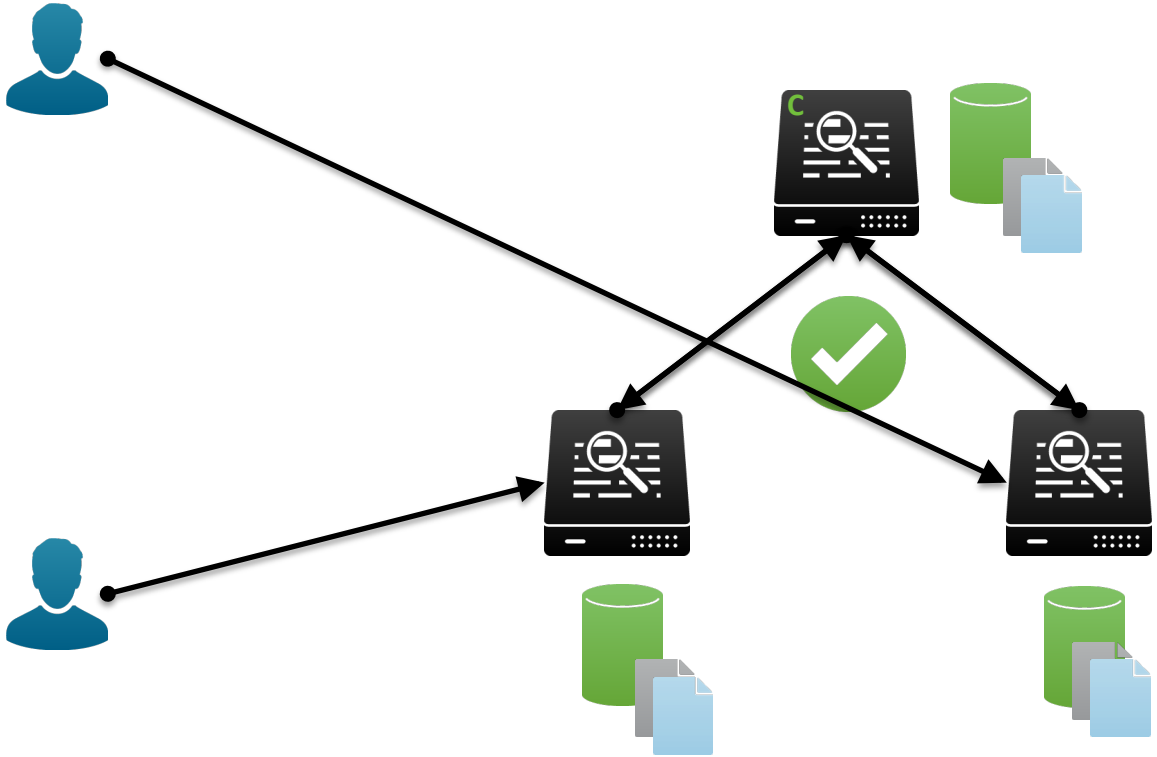
Adhoc search management

- Adhoc search - interactive search run from a user session
- Adhoc searches not replicated
- Captain, however will have global knowledge of all searches.
- GET services/search/jobs will return the global list of searches.
- You can proxy and access adhoc searches from any node.

Reaping of Search Artifacts

- Reaping – Deletion of Search results when TTL (time to live) expires.
- Search Artifacts reaped from the origin node.
- Captain orchestrates reaps of the replicas

Concurrent Changes



Custom App Content

- App devs may "opt-in" their custom configurations for replication under search head clustering
- Example server.Conf from an app would look like:

```
[shclustering]
conf_replication_include.my_custom_file = true
```

UI (comparison)