cisco

Developer Days
Network Services Orchestrator

# Build up NSO LSA with CI/CD

Faster way to deliver complex service(s)

Sudipta Debnath
Software Architect, CX Delivery
10th May 2020

*"Simply put, things always had to be in a production-ready state: if you wrote it, you darn well had to be there to get it running!"*

Mike Miller

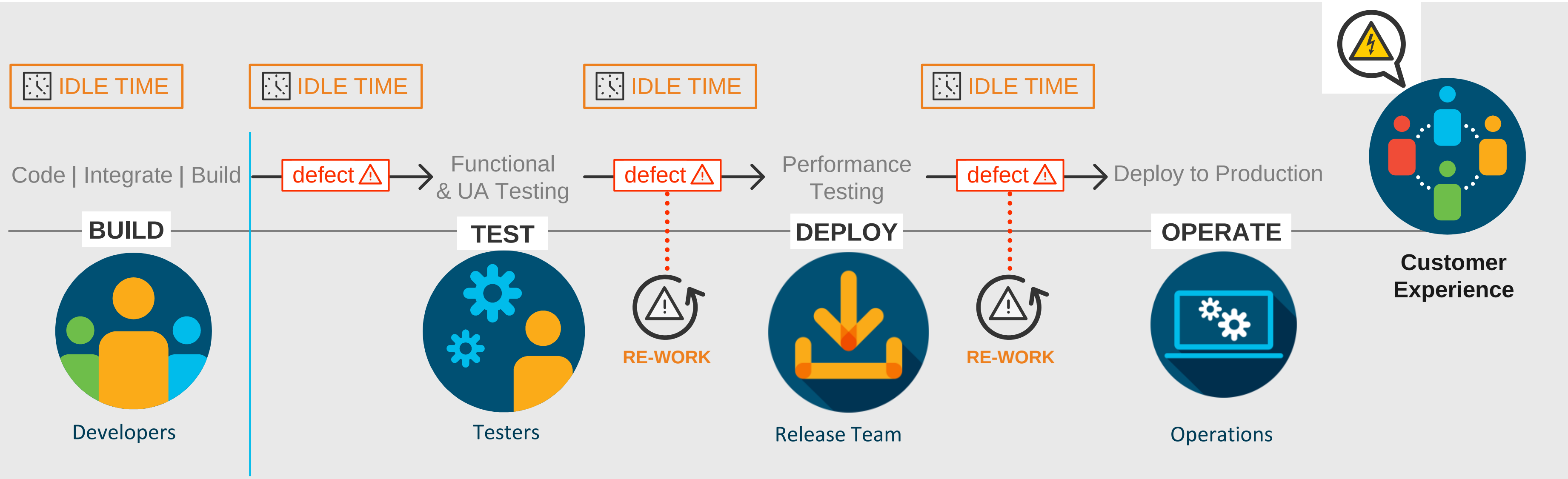
# Challenges – delivering complex software service(s) continuously

Limited Visibility across SW lifecycle

Code Quality Concerns

Manual Process in Complex Environment

Agility & Productivity Issues

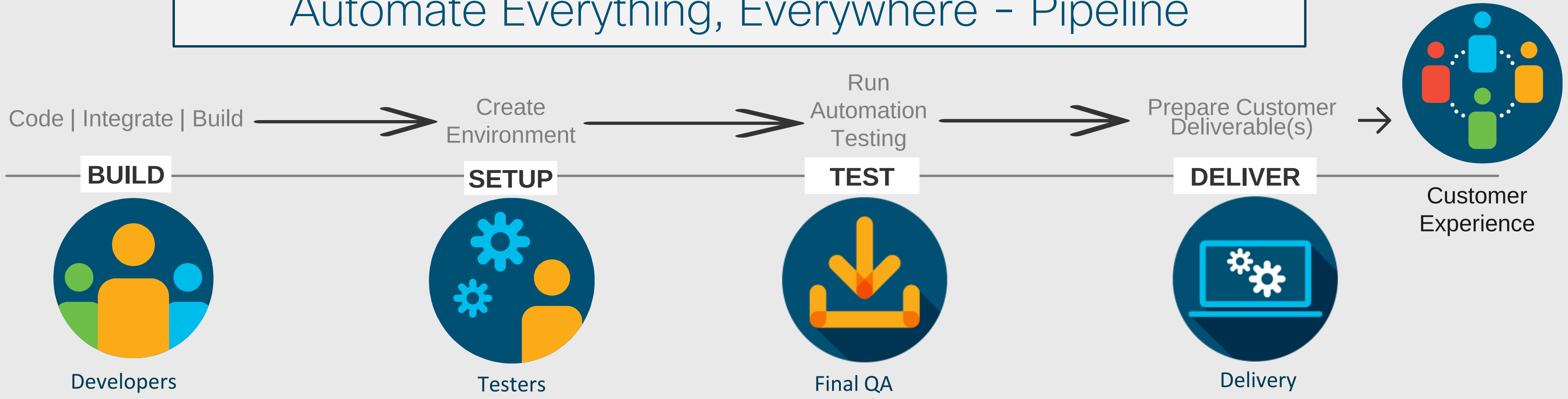


BU

# Continuous end-to-end NSO Delivery automation pipeline

- Agile way to control NSO Delivery
- Create Test environment to mimic actual customer setup
- Run Test Automation & Approve final delivery
- Ease of Use & Efficiency

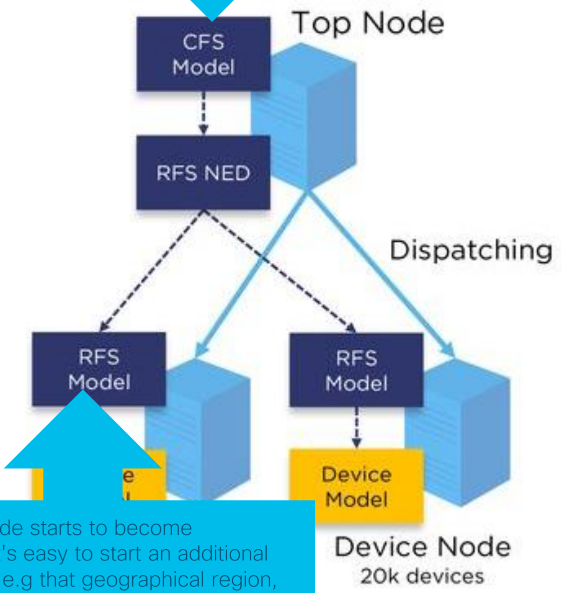
## Automate Everything, Everywhere - Pipeline



The basic idea is to split a service into an upper layer and one or several lower level parts

# NSO LSA – a quick glance

This can be viewed as splitting the service into a customer-facing (CFS) and a resource-facing (RFS) part. The CFS code (upper-level) runs in one (or several) NSO cfs-nodes, and the RFS code (lower-level) runs in one of many NSO rfs-nodes



If one RFS node starts to become overloaded, it's easy to start an additional RFS node for e.g that geographical region, or that data center, thus catering for horizontal scalability at the level of number of managed devices, and number of RFS instances

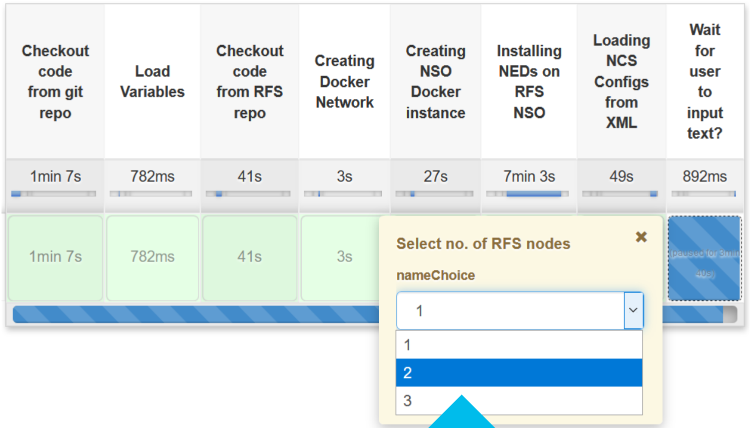
## Why NSO LSA Pipeline is needed?

- An Agile approach with automated Test certification
- Developers can easily test their code without bringing up a heavily built-up infrastructure
- Acceptance of software build can be performed on the fly – no investment in physical infrastructure
- The build quality improvement can be as much as 80% better compared with traditional approaches.

## Factors to be considered – NSO LSA Pipeline design

- Dynamically accept input for creating RFS nodes from Jenkins input
- Create RFS nodes on-the fly no need save any configuration
- Optimize the pipeline by creating all RFS nodes in parallel from Jenkins
- Get hooked up your NSO testing and pull logs out of container before pipeline gets closed

# How Jenkins create LSA dynamically



| Checkout code from git repo | Load Variables | Checkout code from RFS repo | Creating Docker Network | Creating NSO Docker instance | Installing NEDs on RFS NSO | Loading NCS Configs from XML | Wait for user to input text? |
|---|---|---|---|---|---|---|---|
| 1min 7s | 782ms | 41s | 3s | 27s | 7min 3s | 49s | 892ms |
| 1min 7s | 782ms | 41s | 3s | | | | (paused for 3min 49s) |

Select no. of RFS nodes

nameChoice

1

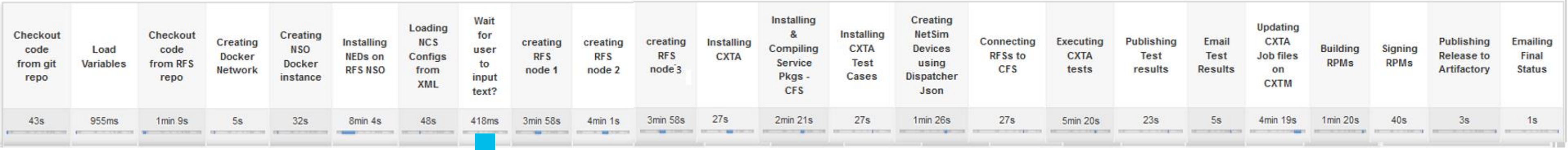
- 1
- 2
- 3

Accepts dynamically number of RFS nodes that user would like to create in order to establish LSA setup

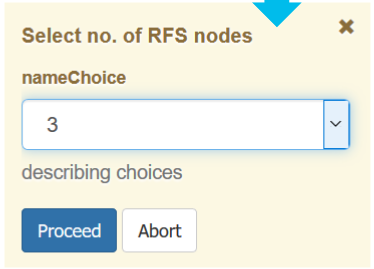
# A Dynamic NSO LSA pipeline in a glance – 3 RFS

3 RFS Nodes

Parallel creation of RFS Nodes



| Checkout code from git repo | Load Variables | Checkout code from RFS repo | Creating Docker Network | Creating NSO Docker instance | Installing NEDs on RFS NSO | Loading NCS Configs from XML | Wait for user to input text? | creating RFS node 1 | creating RFS node 2 | creating RFS node 3 | Installing CXTA | Installing & Compiling Service Pkgs - CFS | Installing CXTA Test Cases | Creating NetSim Devices using Dispatcher Json | Connecting RFSs to CFS | Executing CXTA tests | Publishing Test results | Email Test Results | Updating CXTA Job files on CXTM | Building RPMs | Signing RPMs | Publishing Release to Artifactory | Emailing Final Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 43s | 955ms | 1min 9s | 5s | 32s | 8min 4s | 48s | 418ms | 3min 58s | 4min 1s | 3min 58s | 27s | 2min 21s | 27s | 1min 26s | 27s | 5min 20s | 23s | 5s | 4min 19s | 1min 20s | 40s | 3s | 1s |



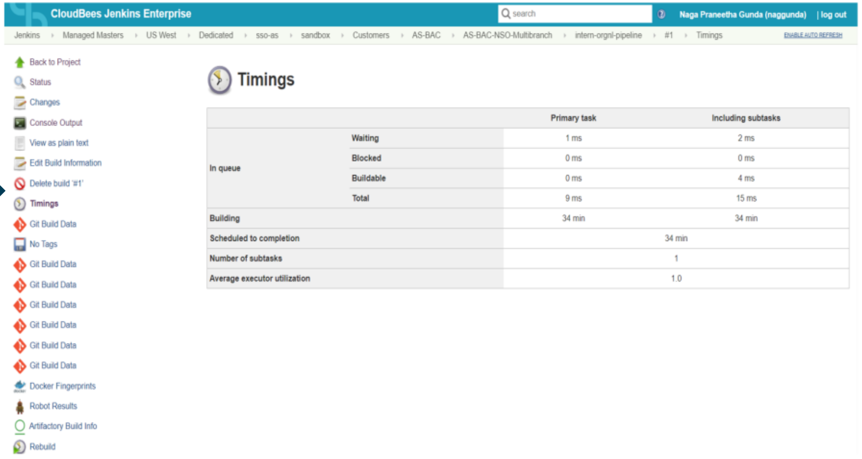
**Select no. of RFS nodes**

**nameChoice**

3

describing choices

Total Execution Time 34 min



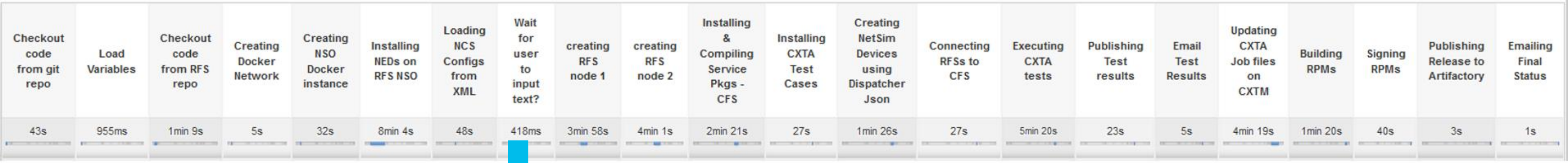
## Timings

| | | Primary task | Including subtasks |
|---|---|---|---|
| In queue | Waiting | 1 ms | 2 ms |
| | Blocked | 0 ms | 0 ms |
| | Buildable | 0 ms | 4 ms |
| | Total | 9 ms | 15 ms |
| Building | | 34 min | 34 min |
| Scheduled to completion | | 34 min | |
| Number of subtasks | | 1 | |
| Average executor utilization | | 1.0 | |

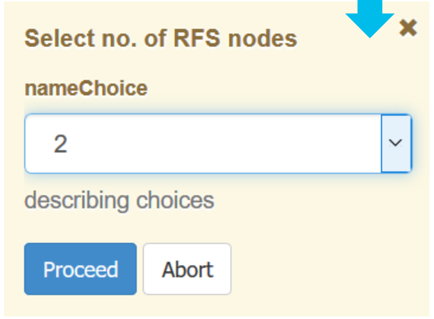
# A Dynamic NSO LSA pipeline in a glance – 2 RFS

2 RFS Nodes

Parallel creation of RFS Nodes



| Checkout code from git repo | Load Variables | Checkout code from RFS repo | Creating Docker Network | Creating NSO Docker instance | Installing NEDs on RFS NSO | Loading NCS Configs from XML | Wait for user to input text? | creating RFS node 1 | creating RFS node 2 | Installing & Compiling Service Pkgs - CFS | Installing CXTA Test Cases | Creating NetSim Devices using Dispatcher Json | Connecting RFSs to CFS | Executing CXTA tests | Publishing Test results | Email Test Results | Updating CXTA Job files on CXTM | Building RPMs | Signing RPMs | Publishing Release to Artifactory | Emailing Final Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 43s | 955ms | 1min 9s | 5s | 32s | 8min 4s | 48s | 418ms | 3min 58s | 4min 1s | 2min 21s | 27s | 1min 26s | 27s | 5min 20s | 23s | 5s | 4min 19s | 1min 20s | 40s | 3s | 1s |



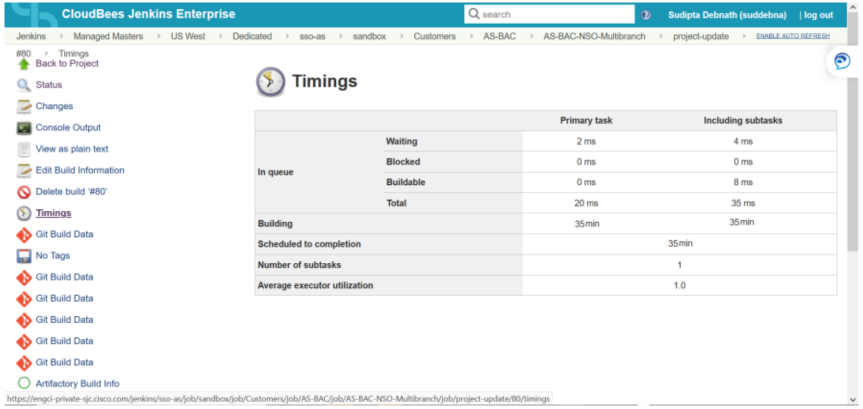
Select no. of RFS nodes

nameChoice

2

describing choices

Total Execution Time 35 min

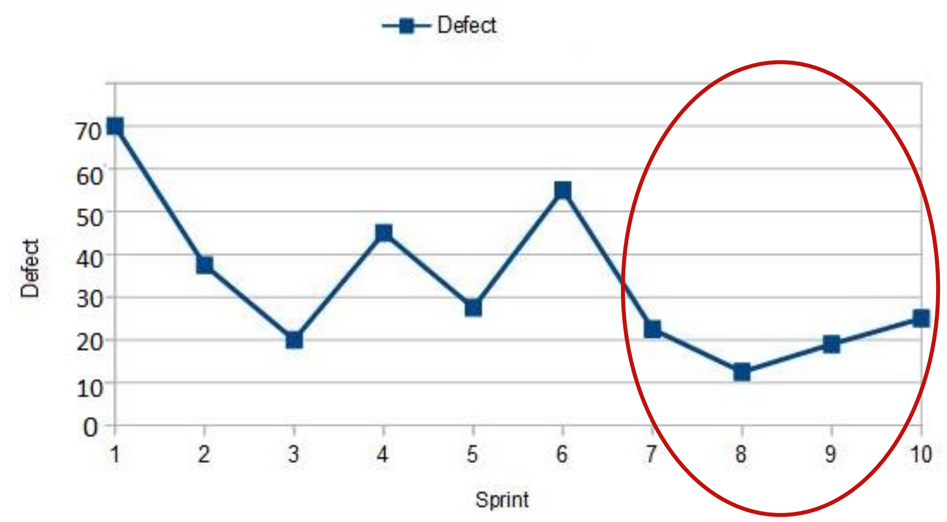


Timings

| | | Primary task | Including subtasks |
|---|---|---|---|
| In queue | Waiting | 2 ms | 4 ms |
| | Blocked | 0 ms | 0 ms |
| | Buildable | 0 ms | 8 ms |
| | Total | 20 ms | 35 ms |
| Building | | 35min | 35min |
| Scheduled to completion | | 35min | |
| Number of subtasks | | 1 | |
| Average executor utilization | | 1.0 | |

https://engci-private-sjc.cisco.com/jenkins/sso-as/job/sandbox/job/Customers/job/AS-BAC/job/AS-BAC-NSO-Multibranch/job/project-update/80/timings

## Change in Defect reported during Acceptance Testing in customer Lab



- Average number of defects came down from 43 (SPRINT 1- SPRINT 6) to 19.25 (SPRINT 7 – SPRINT 10)

## Quality Improvement in Release Delivery

- Lesser number of defects from customer site signifies early detection of defects in development lab
- Defect average trend reduction from 43 to 19 signifies Defect report rate got reduced almost 55%
- Reduction of defect reported by customer signifies quality improvement in Release Delivery.
- Improvement in overall release quality improvement more than 80% (the factor being calculated based on no critical defect reported from UAT, reduction in defect density at UAT, Passed Test coverage during UAT is more than 95%)

# Conclusion

- Dynamic pipeline ensures irrespective of Test suite content remain same there is no additional time it takes to bring up LSA setup on the fly.
- The most easiest way to make your NSO code ready for customer deliverables in a click away.
- Use the setup as an when needed – no dedicated infrastructure.
- You can expect to find more defect(s) prior to shipment- more robust testing is possible.
- Ready to integrate with any standard tool set.