Starlight: Fast Container Provisioning on the Edge and over the WAN

Jun Lin Chen, Daniyal Liaqat, Moshe Gabel, Eyal de Lara
Container Provisioning

► De-facto standard approach for packaging and deploying in cloud
  ➢ Standardized
  ➢ Lightweight
  ➢ Easy to develop and deploy

► Increasingly used outside cloud
  ➢ WAN
  ➢ Mobile
  ➢ Edge

"the edge"
Starlight Contributions

- Container provisioning slow outside datacenter.
- State-of-the-art optimizations make it worse!
- Root cause: design decisions from cloud.
- **Starlight**: accelerator for container provisioning
  - x3 faster, even in cloud
  - Backwards compatible with existing containers, tools, registries, standards.
  - Practically no overhead
  - Open source
Let’s expand on that...
What are Containers?

- **Container** = isolated processes
  - Filesystem, resources

- **Container image** = stack of layers
  - Filesystem is union of layers.

- Easy to develop and package:
  - Start with existing container...
  - ...add new layer on top.

---

Chen, Liaqat, Gabel, de Lara
Deploying Containers on a Node

► Standard 3-phase process for deployment:

1. **PULL**: get compressed layers from registry (container DB)
2. **CREATE**: decompress contents, create mount points.
3. **START**: mount the filesystem and start.

![Diagram of container deployment process](image.png)
Fast Container Provisioning

- Containers-as-a-Service
  - Amazon ECS, Azure Container Instances...

- Function-as-a-Service
  - FaaSNet [Wang et al., ATC’21]

- Security and software updates
  - Log4j

- User mobility
  - [Tiwari et al., HotMobile’19]
Edge Challenges

► High latency, low bandwidth links
  → long downloads

► Limited edge resources
  → no local registry/cache
  → aggressive repurposing

► User mobility
  → frequent reconfiguration
Deploy containers on the edge, measure provisioning time.

- **containerd prov. time** = download + decompress + start + ready for work
- **download**: just download.

Provisioning time grows at a faster rate than download times. When moving to the edge, **prov time triples**. Lower is better.
Containers on the Edge

60%—99% files not needed during startup

*Harter, et al. Slacker: Fast distribution with lazy docker containers, FAST 16*

**eStargz**: state-of-the-art, start containers early, download on-demand

Chen, Liaqat, Gabel, de Lara

---

Fast in cloud, but slow on edge.

Scales badly with RTT

Can be slower than containerd!
Why Slow?

- **Pull-based protocol:**
  - Worker retrieves only layers it needs
  - Multiple long HTTP requests → many roundtrips, queuing
  - On-demand file requests makes this **worse**!
Why Slow?

Layer-based structure:

- Metadata stored per-layer → extra roundtrips
- Cross-layer file duplication → inflates downloads
- Docker Hub study:
  99.4% of files are duplicated
  Zhao et al, CLUSTER '19

Download size (MB)

- Image size
- New files

x10.5 inflation

Chen, Liaqat, Gabel, de Lara
Starlight

► Piecemeal approaches won’t fix core design.
  ➢ ...and we want to be backwards compatible.

► Must rethink deployment pipeline as a whole.

► So that’s what we did with Starlight!

1. Designed new **worker-cloud protocol** (push-based, file-granularity).
2. Implemented components to support it.
Design of Delta Bundle Protocol

▸ Push-based: single request, no roundtrips
▸ Only send what worker needs
▸ **All** metadata before **any** contents
▸ File granularity

<table>
<thead>
<tr>
<th>Header</th>
<th>Body</th>
</tr>
</thead>
</table>

compressed file contents

file list and metadata
Delta Bundle Structure

Image on worker

- #1: C D
- #2: B E A1

my-app:v1

delated

Image requested

- #3: F
- #4: D B F A2

my-app:v2

file renamed

change

contents changed

copy
Delta Bundle Structure

<table>
<thead>
<tr>
<th>File</th>
<th>Hash</th>
<th>Attributes</th>
<th>Source</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>etc/my.cnf</td>
<td>B</td>
<td>-r----------</td>
<td>#2, #4</td>
<td></td>
</tr>
<tr>
<td>etc/ssh</td>
<td>drwxr-xr-x</td>
<td></td>
<td>#2, #4</td>
<td></td>
</tr>
<tr>
<td>etc/sshd_config</td>
<td>A2</td>
<td>-rw-r--r--</td>
<td>#4</td>
<td></td>
</tr>
<tr>
<td>etc/sshd_banner</td>
<td>F</td>
<td>-rw-r--r--</td>
<td>#3</td>
<td></td>
</tr>
<tr>
<td>bin/tar</td>
<td>D</td>
<td>-rw-r--r--</td>
<td>#1, #4</td>
<td></td>
</tr>
<tr>
<td>root/test.txt</td>
<td>F</td>
<td>-rw-r--r--</td>
<td>#3</td>
<td></td>
</tr>
</tbody>
</table>

...
Delta Bundle Structure

Image on **worker**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>C</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>B</td>
<td>E</td>
<td>A1</td>
<td></td>
</tr>
</tbody>
</table>

**my-app:v1**

Image **requested**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#3</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>D</td>
<td>B</td>
<td>F</td>
<td>A2</td>
</tr>
</tbody>
</table>

**my-app:v2**

<table>
<thead>
<tr>
<th>file</th>
<th>hash</th>
<th>attributes</th>
<th>source</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>etc/my.cnf</td>
<td>B</td>
<td>-r----------</td>
<td>#2, #4</td>
<td></td>
</tr>
<tr>
<td>etc/ssh</td>
<td>drwxr-xr-x</td>
<td></td>
<td>#2, #4</td>
<td></td>
</tr>
<tr>
<td>etc/sshd_config</td>
<td>A2</td>
<td>-rw-r--r--</td>
<td>#4</td>
<td></td>
</tr>
<tr>
<td>etc/sshd_banner</td>
<td>F</td>
<td>-rw-r--r--</td>
<td>#3</td>
<td></td>
</tr>
<tr>
<td>bin/tar</td>
<td>D</td>
<td>-rw-r--r--</td>
<td>#1, #4</td>
<td></td>
</tr>
<tr>
<td>root/test.txt</td>
<td>F</td>
<td>-rw-r--r--</td>
<td>#3</td>
<td></td>
</tr>
</tbody>
</table>

...  

**Delta Bundle**

- **Metadata in front**
- **Only new contents**
- **No duplication**

**Starlight: Fast Container Deployment**
Starlight Architecture

- CLI
- StarlightFS
- container
- snapshotter
- Worker
- Cloud
- Starlight protocol (delta bundles)
- Standard protocol
- docker registry
- container
- proxy
- directory DB

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Starlight Operation

Worker

CLI

container

snapshotter

StarlightFS

Cloud

registry

proxy

directory DB

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Starlight Operation

Worker

CLI

PULL my-app

PULL

container

snapshotter

StarlightFS

Cloud

registry

proxy

directory DB

Chen, Liaqat, Gabel, de Lara
Starlight Operation

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB

CLI

container

StarlightFS

Worker

Cloud

registry

have my-app:v1

want my-app:v2

proxy

directory DB
Starlight Operation

CLI

container

StarlightFS

Worker

Cloud

registry

proxy

retrieve contents (layers)

retrieve list of files

directory DB

snapshotter

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Worker

Cloud

Upon receiving a request from the CLI, the snapshotter component in the Worker communicates with the proxy in the Cloud to fetch the compressed layers and metadata for the requested container. The URI for the container is used to retrieve the compressed layers from the registry. The proxy then computes the delta bundle (Δ) list of files and metadata. This list is then fetched and sent back to the Worker, where the delta bundle is computed on the fly, ensuring fast container deployment.

Key Components:
- Worker:
  - CLI
  - Snapshotter

- Cloud:
  - Registry
  - Proxy
  - Directory DB

- Delta Bundle Computed on the Fly

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Starlight Operation

Worker

CLI

Container

Cloud

Registry

Compressed layers

Proxy

List of files and container metadata (SLM)

Directory DB

Snapshottter

StarlightFS

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Starlight Operation

- CLI
- Worker
- PULL finished
- container
- snapshotter
- compressed file contents
- proxy
- Cloud
- compressed layers
- registry
- directory DB
- StarlightFS
- Starlight: Fast Container Deployment
Starlight Operation

CLI
create and start container

container
mount

StarlightFS

Worker

Cloud

registry
compressed layers

proxy

compressed file contents

snapshotter

directory DB

✓ Containers start before download finishes

get SLM for early start

30

Chen, Liaqat, Gabel, de Lara
Starlight Operation

Worker

Cloud

CLI

registry

compressed layers

snapshotter

compressed file contents

proxy

directory DB

container (blocked)

StarlightFS

open pending file (will block)
Starlight Operation

Worker

Cloud

CLI

registry

compressed layers

compressed file contents

proxy

directory DB

snapshotter

container

StarlightFS

container (blocked)
Starlight Operation

- CLI
- Container
- StarlightFS (blocked)
- snapshotter
- payload of pending file
- compressed file contents
- proxy
- compressed layers
- registry
- directory DB

✓ Order optimized to minimize blocking

Worker

Cloud

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Details I Didn’t Discuss

► Generating optimized SLMs
► Starflight filesystem
► Trace collection
► Seekable compressed layers format
► Directory DB
► Downloader and metadata manager
Evaluation

► 21 popular containers
  ➢ 15B+ downloads in Docker Hub
  ➢ Run each until ready

► Controlled deployments
  ➢ tc controls bandwidth, RTT

► Real-world deployment
  ➢ WAN covers 3 continents
Effect of Round-trip Time

- $X = \text{RTT 0 to 300ms, 100Mbs}$
- $Y = \text{resulting provisioning time}$

- containerd v1.5.0

![Graph showing the relationship between RTT and Fresh Provisioning time](chart.png)
Effect of Round-trip Time

- **X** = RTT 0 to 300ms, 100Mbs
- **Y** = resulting provisioning time
  - containerd v1.5.0
  - eStargz v0.6.3: pull-based, on-demand
Effect of Round-trip Time

- $X = \text{RTT 0 to 300ms, 100Mbs}$
- $Y = \text{resulting provisioning time}$

- containerd v1.5.0
- eStargz v0.6.3: pull-based, on-demand
- Download optimized delta bundle
  - Lower bound without early start

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Effect of Round-trip Time

- X = RTT 0 to 300ms, 100Mbs
- Y = resulting provisioning time
  - containerd v1.5.0
  - eStargz v0.6.3: pull-based, on-demand
  - Download optimized delta bundle
    - Lower bound without early start
  - Starlight
Effect of Round-trip Time

**Starlight:**
- Fastest (even with cloud RTT)
- Scales well with latency.
- Outperforms `wget`.
How Much Faster Are We?

**speedup over containerd**
(harmonic mean of 21 containers)

- Fresh deployment
  - ✓ Outperforms containerd 3x, estargz 1.9x
  - ✓ Faster than wget
  - ✓ Scales better with RTT.
  - ✓ Extremely fast updates: 4—5x compared to fresh

- Update

---

Chen, Liaqat, Gabel, de Lara

Starlight: Fast Container Deployment
Runtime Overhead?

- 150ms, 100Mbs
- YCSB Workload A
- In-memory and on-disk database.

No overhead:
✓ Workers start earlier, finish faster.
✓ Same peak performance.
Additional Experiments

► On WAN → similar results.
► In cloud → Starlight fastest.
► Bandwidth
► Scalability
► Proxy memory and runtime
► Detailed case analysis
Conclusion and Future Work

► Container provisioning is slow.
  ➢ Layered structure
  ➢ Pull-based protocols

► Starlight: new provisioning protocol, filesystem, storage format.
  ✓ Faster deployment on edge, cloud
  ✓ Backwards compatible, transparent
  ✓ No overhead
  ✓ Open source

► Future work:
  ➢ Collect traces online
  ➢ Predict/learn file order
  ➢ Jointly optimize multi-container deployments

https://github.com/mc256/starlight
mgabel@cs.toronto.edu