Singularity - Container für die HPC Welt

Egbert Eich
Project Manager HPC
SUSE Linux GmbH/eich@suse.com

Über mich …

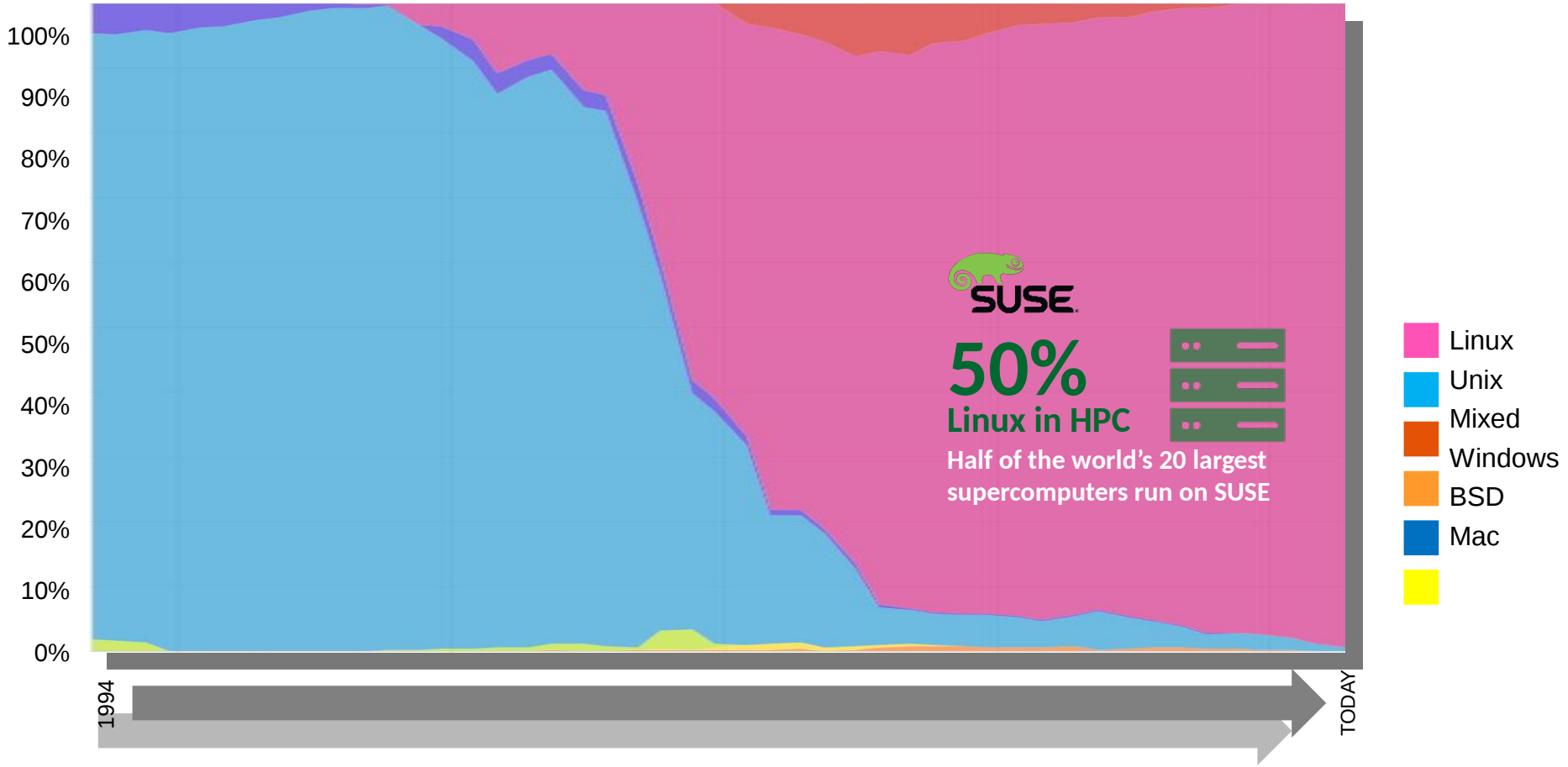
- **UNIX Betriebssysteme seit 1991 (AIX, Digital UNIX, …)**
- **Seit 25 Jahren Linux**
- **Seit 2000 bei der SUSE Linux GmbH**
  - Entwickler/Maintainer in den SUSELabs zuständig für den Graphic Stack - DRM, Mesa, DRI, X Window System, (Wayland)
  - Seit August 2016 Project Manager und Architekt für High Performance Computing

HPC bei SUSE

- SUSE Linux **Enterprise** Server läuft auf ca 50% der Top20 Supercomputer
- Bei denTop500 Marktanteil von 13.4%

13.4% SUSE OS Share, including Cray Linux Environment

Linux on 99.4% of the Top 500 Supercomputers



Neu: HPC Modul für SLES 12

- **HPC Modul für SUSE Linux Enterprise Server 12 (seit SP2)**
  - HPC user space Stack:
    - Workload Manager (SLURM)
    - Parallele shells
    - Cluster Management Tools
    - Performance Measurement Tools
    - Library Module System
    - Numerische Bibliotheken (seriell und parallel)
    - MPI Bibliotheken
    - Bibliotheken für HPC File Formate
  - Angelehnt an Software-Auswahl des OpenHPC Projekts (http://openhpc.community)
  - Zielgruppe: Mittlere Unternehmen mit HPC Bedarf
- **SUSE „Factory First“ Policy:**
  - Gleiche Software-Pakete auch in openSUSE Factory & Tumbleweed sowie in openSUSE Leap

Wiederkehrendes Problem im HPC Umfeld

- **Unterschiedliche User benutzen**
  - unterschiedliche Versionen der gleichen Bibilothek
  - unterschiedliche Varianten der gleichen Version einer Bibliothek unterscheiden sich zB in der
    - verwendeten MPI Implementierung
    - Compiler / Compiler-Version
    - Compiler-Optimierung
- **Folge: Applikation läuft auf der Entwicklungs-Workstation, auf dem Cluster jedoch nicht: fehlende Bibliotek oder falsche Version.**

Lösungen

Statisch gelinkte Applikationen

- Nachteil: u.a. Verteilung von Bug-Fixes schwierig.

Library Module

- Parallele Installation aller unterschiedlicher Varianten und Versionen
  - gleiche Namen, unterschiedliche Verzeichnisse `/usr/lib/<compiler-variante>/<mpi-variante>/<library-version>/libFoo.so`
  - Modifizieren des Environments beim Einloggen, sodass die gewünschten Bibliotheks-Versionen und -Varianten beim Bauen und zur Laufzeit gefunden werden.
  - Wird vom HPC Modul für SLES unterstÜtzt, d.h. SUSE kümmert sich um Paketierung der unterschiedlichen Bibliotheksversionen und Varianten, stellt passende Modul-Control Informationen zur Verfügung und erlaubt die parallele Installation.
- Nachteile:
  - Anwendung nicht Portabel - läuft nur, wenn die richtigen Bibiliotheken auf dem Zielsystem vorhanden sind.
  - Admins haben keinen Überblick, welche Bibliotheks-Versionen noch in Verwendung sind.

Container - Bessere Lösung?

- Ausweg aus dem Abhängigkeits-Dilemma:
  Jeder Benutzer erhält 'seinen' Container mit genau 'seiner' Build- und Laufzeit-Umgebung
- Portabel - Container funktioniert immer und überall gleich:
  - enthält alle Komponenten, die für den vorgesehenen Anwendungs-Workflow notwendig sind.
  - unabhängig von der Version und dem Flavor des Zielsystems
  - unabhängig von der Konfiguration des Zielsystems
  - Ergebnisse sind weltweit reproduzierbar
  - Einfach verteilbar

Anforderungen an das Container-System

- Keine besonderen Rechte notwendig zum starten von Containern
- Einfache Portierbarkeit auf verschiedene Gast-Systeme
- Einfache Integrierbarkeit in existierende Infrastrukturen
- Benutzer kann eigene Container erstellen und verteilen
- Funktioniert mit allen Workload Managern

Docker

- benötig root-daemon zum Starten von Containern: funktioniert nicht mit gängigen Workload Managern.
- läuft nicht auf allen Gast-Systemen

Singularity

Container System zugeschnitten auf die Bedürfnisse von HPC Usern:

- **Unterstützt nativ GPU, InfiniBand und MPI**
- **Sehr niedriger startup-overhead**
- **Funktioniert mit gängigen Resource Managern (SLURM, Torque etc).**

Weitere Features

- Erlaubt Zugriff auf user-eigene resourcen (home)
- Erlaubt den Import (bind-mount) weiterer Pfade des host-systems
- Bind-monted /dev in den Container: Zugriff auf host devices
- Bind mounted /etc/resolv.conf, /etc/hosts, ... in den Container
- Virtualisiert mount namespace: Container Image und mounts nur sichtbar innerhalb des Containers
-

Weitere Features (cont.)

- **Virtualisiert keinen network namespace: gleiche Netzwerkkonfiguration im Container wie ausserhalb - keine zusätzliche Netzwerkkonfiguration notwendig.**
- **Unterbindet root-Escalation innerhalb des Containers**
  - Blockiert SUID/SGID bit innerhalb des Containers,
  - Setzt Capabilities zurück,
  - user 'foo' bleibt user 'foo' im Container

Voraussetzung

- **Betriebsystem – eine Linux-Variante**

Container Image

- **Verwendet einen einzelnen .img file.**
  - Vorteilhaft auf verteilten, parallelen Filesystemen
  - Zugriff geregelt durch normale Filesystem Permissions
  - Platzsparend: Image ist sparse Datei
  - Kein caching des Containerinhalts
- **Weitere supportete Formate:**
  - Directory
  - tar[.gz|.bz2]
  - cpio[.gz]
- **Supportete URIs:**
  - http://|https:// Lade .img von angegebener URI
  - docker:// Lade ein Docker Image von einer Registry (konfigurierbar)
  - shub:// Lade .img file von Singularity-Hub

Ablauf

- **Erzeuge Image:**

```
$ sudo singularity create [image]
```

- **Bootstrap image**

```
$ sudo singularity bootstrap [image] [definition.def]
```

- **Führe image aus:**

```
$ singularity shell [image]
$ singularity exec [image] [/pfad/zum/programm]
$ singularity run [image]
$ ./image
```

Ablauf (2)

- **stdin/stout/sterr der applikation im Container werden an die ausführende shell weitergeleitet, d.h. singularity funktioniert in der pipe:**

```
$ singularity exec /tmp/Demo.img xterm
$ singularity exec /tmp/Demo.img python script.py
$ singularity exec /tmp/Demo.img python < /path/to/python/script.py
$ cat /path/to/python/script.py | singularity exec /tmp/Demo.img python
```

- **Auch MPI Programme funktionieren:**

```
$ mpirun -np X singularity exec /pfad/zum/container.img /usr/bin/mpi_program_im_Container
```

Wie funktioniert das?

- **Operationen, wie mknod, mount, chroot benötigen erweiterte Rechte Preis dafür, dass ein 'normaler' Benutzer singularity ausführen kann:**
  - SUID root Binary

    Absicherung:
    - statische gelinktes Binary
    - Prüfung, ob config Datei root gehört und nur von root geschrieben werden kann.
    - Setzen der eUID auf die calling UID
    - Auditierbarkeit:
      - Setze eUID zu 0 nur wenn nötig
      - Setze Rechte sofort anschliessend wieder zurück
    - Setzt MS_NOSIUD und O_CLOEXEC flags.
- **SUID Root ist ein Problem für das Security Team bei SUSE**
  - User Namespaces? Capabilities?

Weiteres Problem ...

- **Loopback-Mount von fast 'beliebigen' Images durch normale Benutzer auf multi-user Systemen:**
  - Linux Filesystem Code ist nicht robust gegen korrupte oder bösartige Filesysteme.
- **Abhilfe?**
  - Singularity kann erlaubte Images auf solche, die dem Singularity-User gehören beschränken.


SUSE®
We adapt. You succeed.