



The PHP Company

Job Queue in Zend Server 5.0

Jan Burkl
System Engineer, Zend Technologies

Agenda

- Was ist Job Queue und wofür ist sie gut?
- Wie sie funktioniert
- Verwendung der API
 - Jobs erstellen, Parameter weiterleiten
 - Parameter akzeptieren
 - Zeitplan und wiederkehrende Jobs
 - Failure und Success Reporting
- Quick Tour des Administration Interfaces
- Zum Nachtisch: ein paar Benchmarks
- Für die, die noch nicht genug haben...

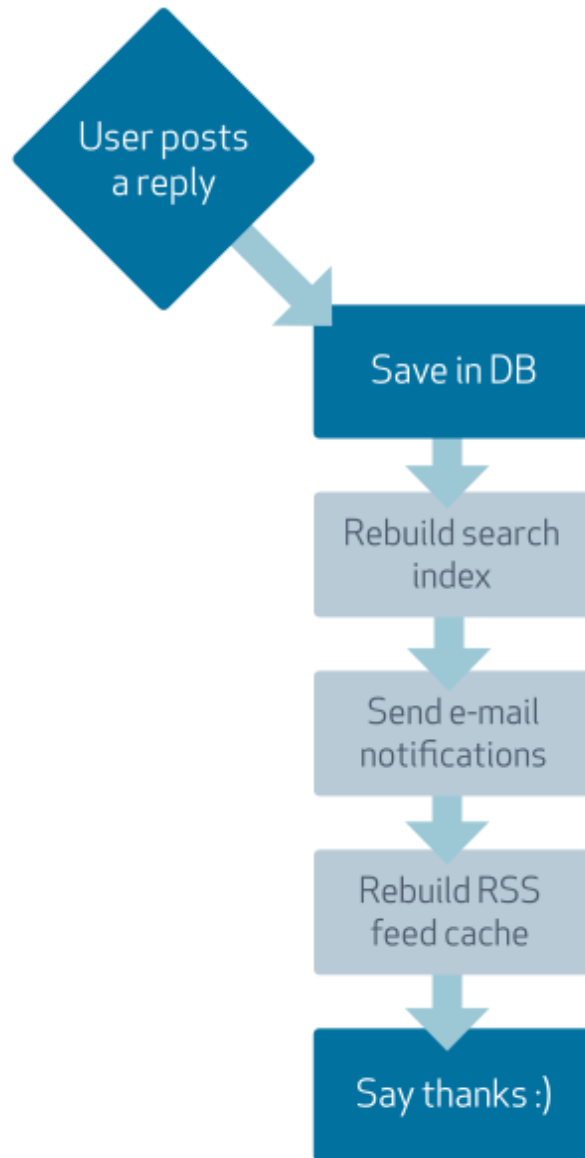
Was ist Job Queue?

...und wofür ist sie gut?

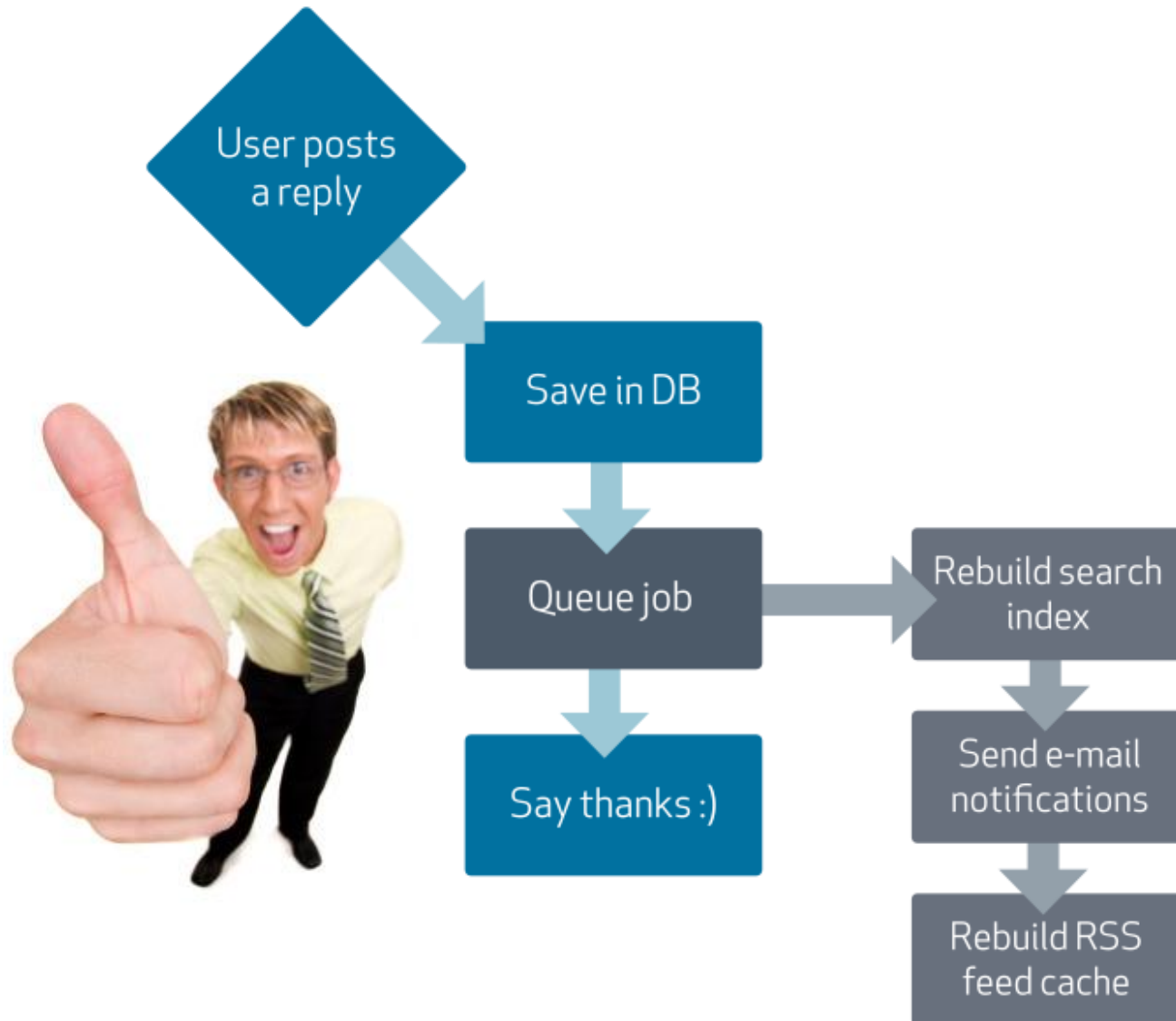
Was versuchen wir zu lösen?

- Off-line Bearbeitung?
 - ▶ Web Applikationen tendieren dazu in HTTP Request/Response Cycles zu “leben”
 - ▶ Was ist zu tun, um Vorgänge Off-line zu nehmen?
 - ▶ Was ist zu tun, wenn periodische Ausführung benötigt wird?
- Es ist unter anderem eine Frage von User-Experience:
 - ▶ User warten lassen ist nicht hilfreich
- Zend Server Job Queue erlaubt Ihnen Vorgänge Off-line auszuführen!
 - ▶ Dinge asynchron, später oder auf einem anderen Server laufen zu lassen
 - ▶ Dinge periodisch laufen zu lassen

Beispiel: Ein On-line Forum



Und jetzt mit Job Queue!



Job Queue erlaubt Ihnen...

- bestimmte Tasks in eine separate Execution Queue zu packen
 - ▶ Off-load zu einem späteren Zeitpunkt (oder sogar parallel laufen lassen)
 - ▶ Off-load auf einen anderen Server
- bestimmte Tasks zu einem definierten Zeitpunkt auszuführen
 - ▶ Verschieben des Processing Load in Off-Hours
- bestimmte Tasks periodisch auszuführen
- Während...
 - ▶ Große Mengen von bestehender Infrastruktur & Code wiederverwendet werden können
 - ▶ Sichergestellt wird, dass nichts verloren geht
 - ▶ Alles von einer PHP API erledigt wird

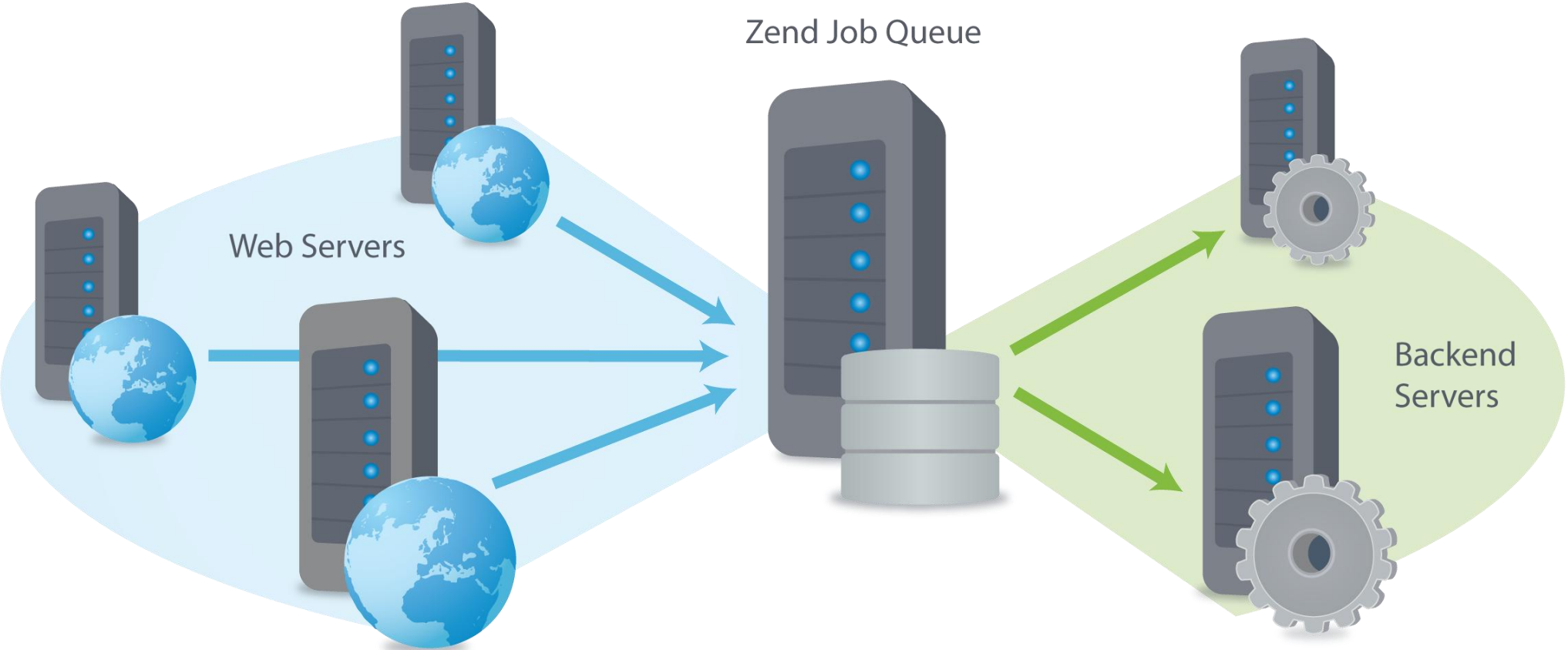
Also, es ist ein hübscher Cron?

- Nein
 - ▶ Sie können Dinge **JETZT** ausführen, jedoch ohne auf das Ende warten zu müssen
 - ▶ Sie können Dinge **einmalig** ausführen, nicht zwingend sofort
 - ▶ Sie können Dinge **periodisch** ausführen (wie cron)
 - Und haben volle Kontrolle über sie - Starten, Stoppen, Verschieben, Fortsetzen von der PHP API
 - ▶ Job Queue gibt Ihnen genaue Übersicht über was gerade passiert
 - Erhalten Sie Benachrichtigungen über Failed Jobs, analysieren Sie Errors und Re-queue
 - Behalten Sie den Überblick über vergangene, laufende und wartende Jobs in der GUI
 - API für Querying Job Status und Failure Handling
 - ▶ Keine Hacks notwendig

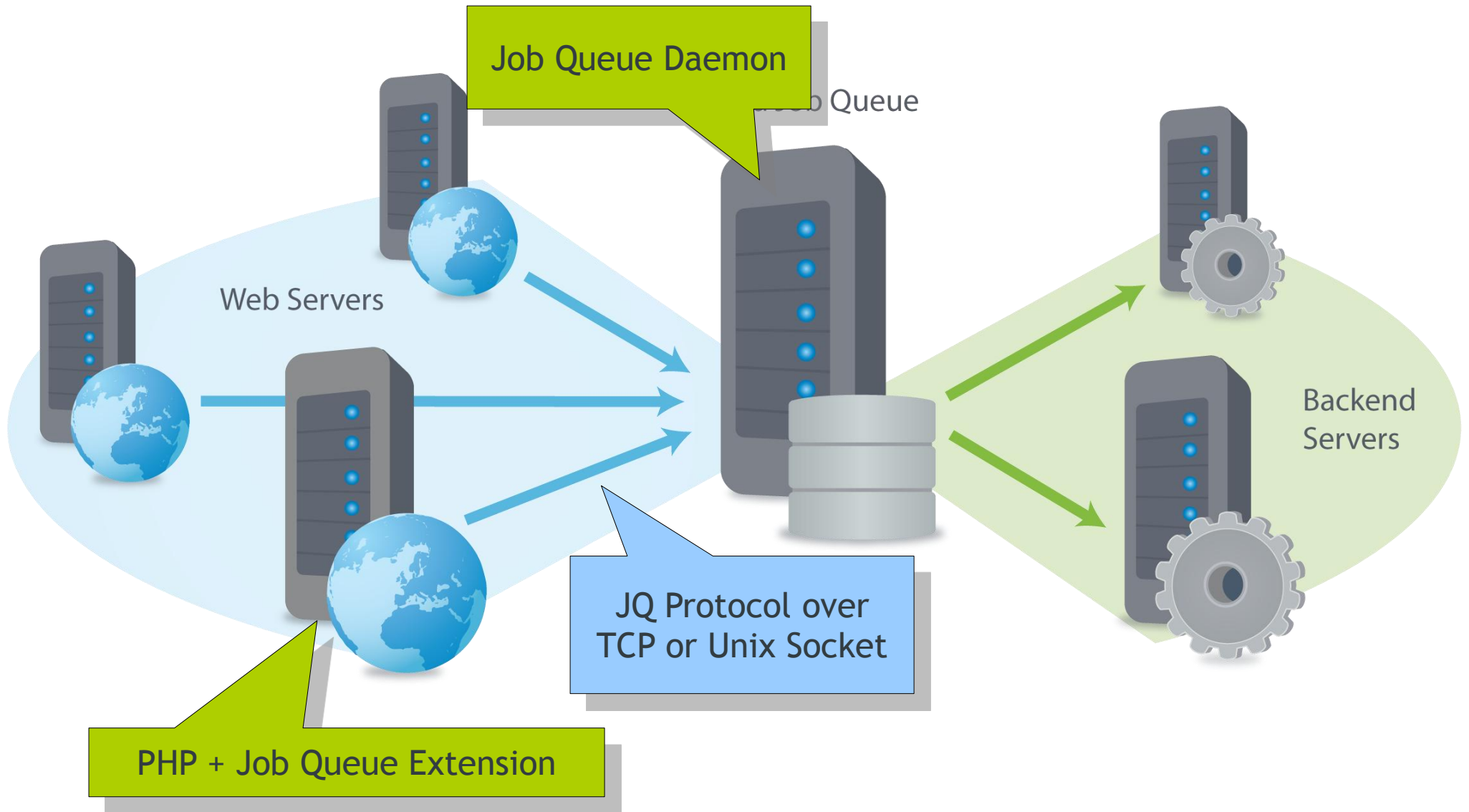
Wie genau funktioniert es?

Architektur und ein paar Interna

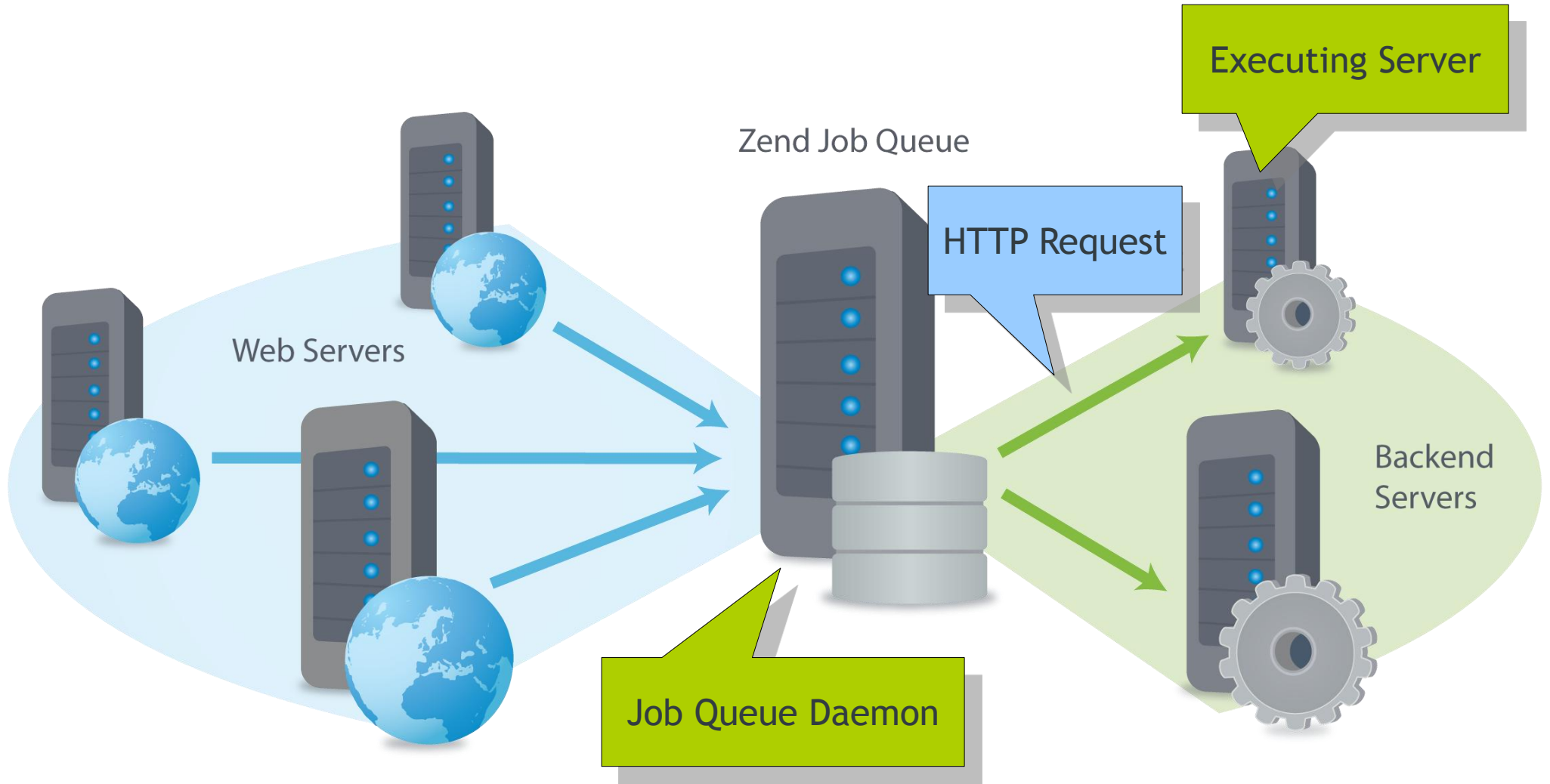
Job Queue 4.x - Architekturüberblick



Job Queue 4.x - Architekturüberblick



Job Queue 4.x - Architekturüberblick



Verwendung der Job Queue API

Kreieren & Ausführen von Jobs

Die *ZendJobQueue* Class

- Die *ZendJobQueue* Klasse beinhaltet fast die komplette PHP API für Job Queue
- Um Tasks auszuführen, müssen Sie sich mit einem Job Queue server verbinden, in dem Sie ein *ZendJobQueue* Object instantieren :

```
// Connect to the default JQ server  
$queue = new ZendJobQueue ();
```

```
// Connect to any other JQ server  
$queue = new ZendJobQueue ("tcp://1.2.3.4:5678");
```

Jobs kreieren

- Jobs werden durch Verwendung der `createHttpJob()` Methode kreiert
- ```
$queue = new ZendJobQueue ();
$queue->createHttpJob (
 'http://backend.local/jobs/somejob.php');
```
- 
- Weitergabe eines Pfads anstatt einer kompletten URL kreiert den Job mit `$_SERVER['HTTP_HOST']` im Host Name

```
$jobPath = '/jobs/otherjob.php' ;

$queue->createHttpJob ($jobPath) ;
// This is equivalent to:
$queue->createHttpJob ('http://' .
 $_SERVER['HTTP_HOST'] . $jobPath) ;
```

# Parameter Weitergabe

---

- Einfache Parameter können als Part eines Query Strings übergeben werden
  - ▶ Dies ist verfügbar innerhalb des Jobs in `$_GET`
- Komplexe Parameter können im 2. Parameter des `createHttpJob()` übergeben werden
  - ▶ Übergabe eines assoziativen Arrays key => value Paaren
  - ▶ Wert kann beliebige Daten beinhalten, die auch mittels JSON repräsentiert werden können
    - Null, Booleans, Strings, Integers, Floating Point Zahlen
    - Indizierte Arrays (einschließlich Nested Arrays)
    - Objekte und assoziative Arrays

# Parameterübergabe (Beispiel)

---

```
$params = array(
 'cart' => array(
 'items' => array(
 array('id' => 324, 'qty' => 1, 'price' => 19.95),
 array('id' => 75, 'qty' => 2, 'price' => 14.95,
 'size' => 'XL')
),
 'total' => 49.85,
 'coupon' => null,
 'giftwrap' => true
),
 'user' => $user
);
```

```
$queue->createHttpJob(
 'http://backend/jobs/checkout.php', $params);
```

# Parameter-Zugriff

---

- Im Job code, verwenden Sie die statische `ZendJobQueue::getCurrentJobParams()` Methode:

```
$params = ZendJobQueue::getCurrentJobParams ();
var_export ($params);
/* Output will be:
array (
 'cart' => array (
 'items' => array (
 0 => array (
 'id' => 324,
 'qty' => 1,
 'price' => 19.95,
),
 ...
),
 ...
*/
```

- Sie können auch `json_decode()` des Raw POST Bodys verwenden:

```
$params = json_decode (file_get_contents ('php://input')) ;
```

# Weitere Job Optionen

---

- Der 3. Parameter des *createHttpJob* ist ein assoziatives Array von Optionen:
  - ▶ *name*
  - ▶ *priority*
  - ▶ *persistent*
  - ▶ *predecessor*
  - ▶ *http\_headers*
  - ▶ *schedule*
  - ▶ *schedule\_time*

# Deferred Jobs kreieren

---

- Execution Time über *schedule\_time*:

```
// Process the form at 3:00 am
$runAt = date('Y-m-d h:i:s',
 strtotime('+1 day 3:00am'));
$options = array(
 'schedule_time' => $runAt
);

$queue->createHttpJob('http://backend/jobs/process.php',
 $_POST, $options);
```

- Der Job wird nicht vor der bestimmten Zeit ausgeführt
  - ▶ Abhängig vom Queue Load, wird es eventuell danach ausgeführt

# Recurring Jobs kreieren

---

- Recurring Jobs durch *schedule* Option.
- cron-ähnliche Syntax, die das Scheduling spezifiziert

```
// Run on Sunday, Monday, and Tuesday at midnight
$jq->createHttpJob(
 'http://localhost/jobs/feed/405',
 array(),
 array('schedule' => '0 0 * * 0,1,2'));
```

```
// Run every other day of the month at 2:30pm
$jq->createHttpJob(
 'http://localhost/jobs/feed/405',
 array(),
 array('schedule' => '30 14 */2 * *'));
```

# Reporting von Logical Failures

---

- Ein logischer Fehler tritt auf, wenn etwas bei der Programmierung scheitert
  - ▶ Z.B. Versand von Emails oder Verbindung zu SOAP Service schlägt fehl
  - ▶ Unterschied zu einem Execution Failure, der eine technische Störung darstellt (Z.B. TCP Error oder HTTP 500 Error vom Server)
- Logische Fehler müssen programmatisch gemeldet werden:

```
if (! $pp->sendPaymentRequest()) {
 // Report failure
 ZendJobQueue::setCurrentJobStatus(
 ZendJobQueue::FAILED,
 "Error from payment service: " . $pp->getError());
}

// Everything went well
ZendJobQueue::setCurrentJobStatus(ZendJobQueue::OK);
```

# Quick Tour durch UI

# Demo Time!

The screenshot shows the Zend Server web interface in a browser window. The address bar displays "http://il-pm1.zend.net:10081 - Zend Server". The interface includes a navigation menu with "Monitor", "Rule Management", "Server Setup", and "Administration". Below this, there are sub-menus for "Dashboard", "Events", "Jobs", "Queue Statistics", "Code Tracing", "Server Info", "PHP Info", and "Logs". A filter dropdown is set to "Failed Jobs", and a "Show Filter Details" link is visible. The main content area shows a table of failed jobs with the following data:

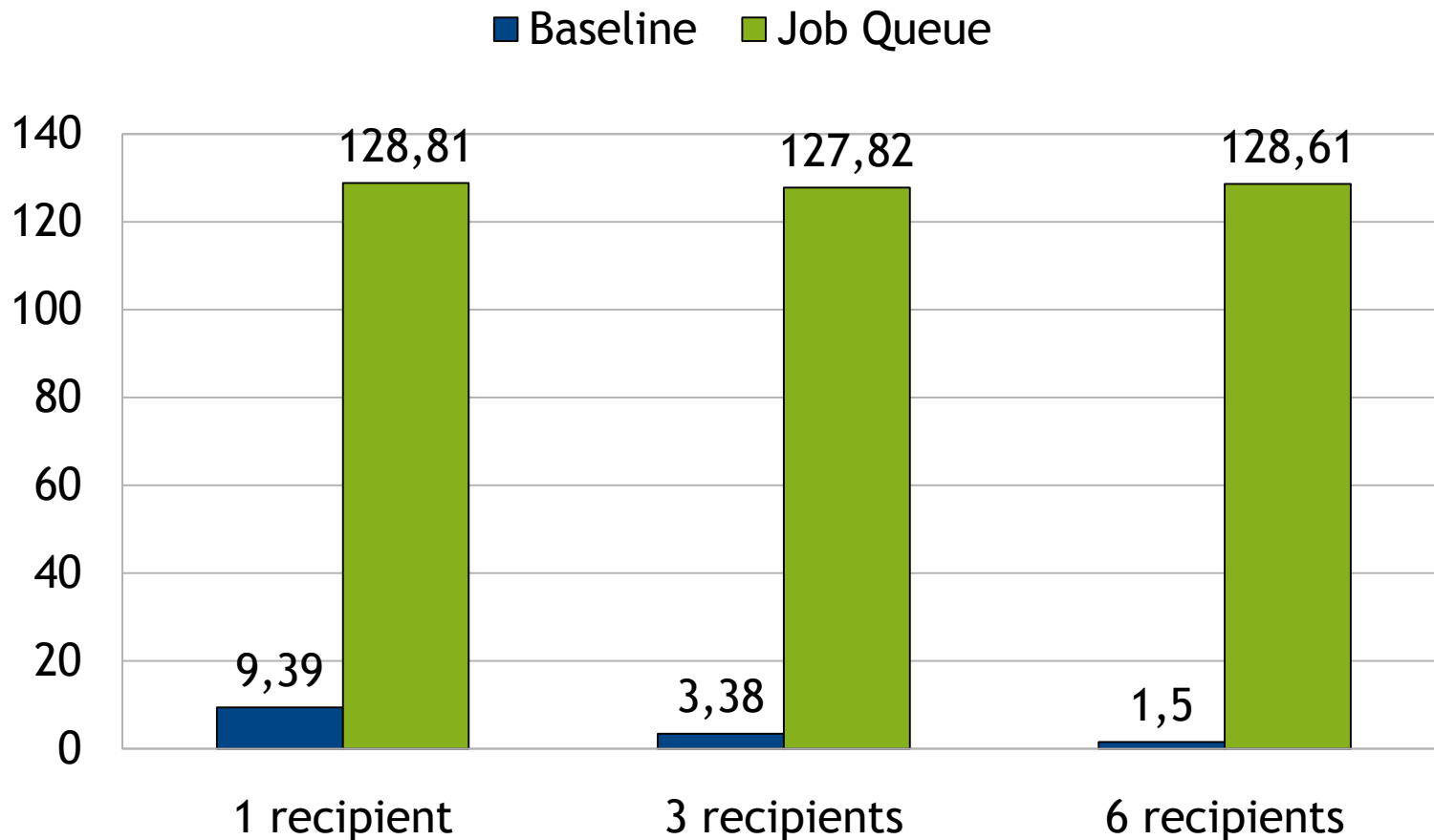
| ID    | URL                                  | Application | Status | Priority | Run Time     |
|-------|--------------------------------------|-------------|--------|----------|--------------|
| 21051 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 14:35 |
| 21050 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 13:35 |
| 21049 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 12:35 |
| 21048 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 11:35 |
| 21047 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 10:35 |
| 21046 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 09:35 |
| 21045 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 08:35 |
| 21044 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 07:35 |
| 21043 | http://backnet.intra/jobs/report.php |             | failed | normal   | 28-Oct 06:35 |

At the bottom of the table, there is a "Delete" button and navigation controls showing "1 out of 32". A "Restart PHP" button is located at the bottom right of the interface.

# Ein paar Benchmarks

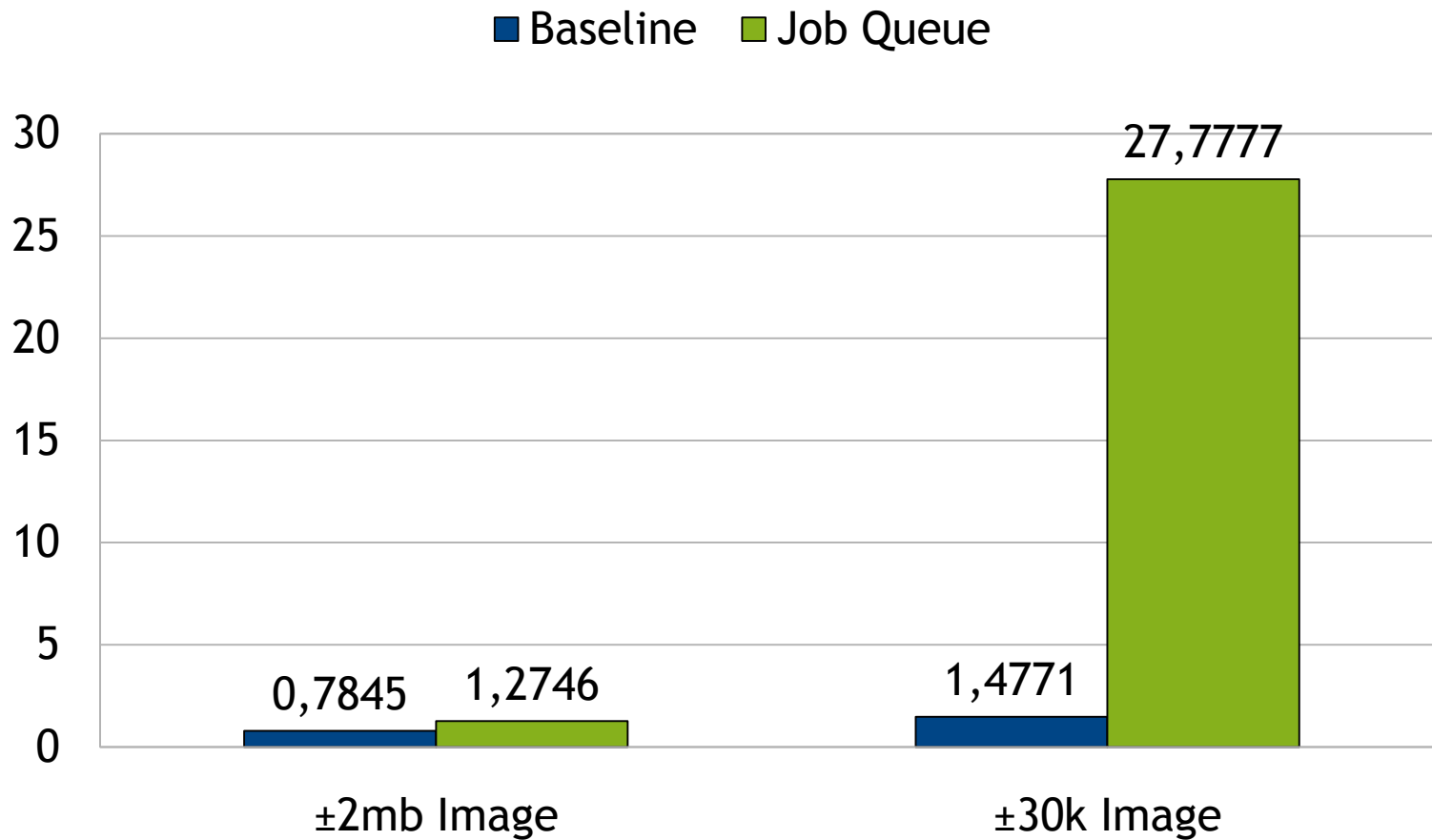
# Emails versenden

- Requests/Second bei einem einfachen E-mail Versand-Script:



# Image Konvertierung

- Image Konvertierung mit Hilfe von ImageMagick



# Final Words

Wie geht's weiter?

# There's more!

---

- Prioritäten managen
- Job Dependencies
- Querying nach Jobs
- Job Status und Queue Statistics überprüfen
- Recurring Jobs & Queues Verschieben/Fortsetzen
- Custom HTTP Headers weitergeben
- Failure Handling und Retry Control
- Load Management und Balancing

Vielen Dank!

# Using the Job Queue API

Querying for jobs and checking status

# Querying for job status

- Wenn die Job Id bekannt ist, kann der Status überprüft werden:

```
// When creating the job:
$jobId = $jq->createHttpJob($url, $params, $options);

// To check for the job's status (must be connected to the
same queue)
$status = $jq->getJobStatus($jobId);
var_export($status);

// Output is:
array (
 'id' => 126,
 'type' => 1,
 'status' => 4,
 'priority' => 1,
 'persistent' => false,
 'script' => 'http://localhost/mail/job.php',
 'vars' => '{"email":"shahar.e@zend.com"}',
 'output' => 'HTTP/1.1 200 OK ...<snip>',
 'creation_time' => '2009-10-21 19:40:01',
 'start_time' => '2009-10-21 19:41:03',
 'end_time' => '2009-10-21 19:41:05',
);
```

# Querying for jobs

---

- Die Job List kann nach unterschiedlichen Parametern durchsucht werden:

```
// Query for all failed "Sales Report" jobs
$jobs = $jq->getJobsList(array(
 'status' => ZendJobQueue::STATUS_FAILED,
 'name' => 'Sales Report'
));

// Try to re-run failed jobs
foreach($jobs as $job) {
 echo "Attempting to re-queue failed job #{$job['id']}...\n";
 $jq->restartJob($job['id']);
}
```