Monitoring Payment Queues

Łukasz Lipski

IT Infrastructure Specialist, Nordea IT Polska

NORDEA IT POLAND

- HQ in Gdynia, Poland, with 300 employees
- supporting the banking systems in Latvia, Lithuania and Estonia
- providing IT services for our Nordic colleagues in Nordea Bank AB

ZABBIX @ NORDEA

ZABBIX

- currently 2.4.5
- 350 monitored hosts, 40k items,
  20k triggers and nvps below 1k
- Oracle as the database backend (11.2)
- primary monitoring system among others
  (Oracle EM, SCOM, Dynatrace, ...)
- key systems: core banking, card traffic, e-banking

MONITORING PAYMENT QUEUES

- short talk
- focusing on one product, partially applicable to others
- “the most you can do with the least effort”

IBM MQ – INTRODUCTION

- IBM's messaging solution since 1992
- the name: MQM, MQSeries, Websphere MQ, IBM MQ since 2014
- the product: largely the same for the last 20 (sic!) years
- quite popular, quite reliable, a bit strange

BASIC STUFF

- MQ filesystem space
  (usually /var/mqm)
- global error reports
  (.FDC files under /var/mqm/errors)
- Queue Manager status
  and local QM error reports
- status of the listener process (runmqlsr)
- channel status

BASIC STUFF – CONT.

Queue Manager status

```
dspmq

echo "ping␣qmgr" | runmqsc QMNAME
```

Listener status

```
ps -ef | grep runmqlsr

echo "display␣lsstatus()" | runmqsc QMNAME
```

Channel status

```
echo "display␣chstatus()" | runmqsc QMNAME
```

QUEUE MONITORING – BASIC METRICS

Number of messages on queue: current depth

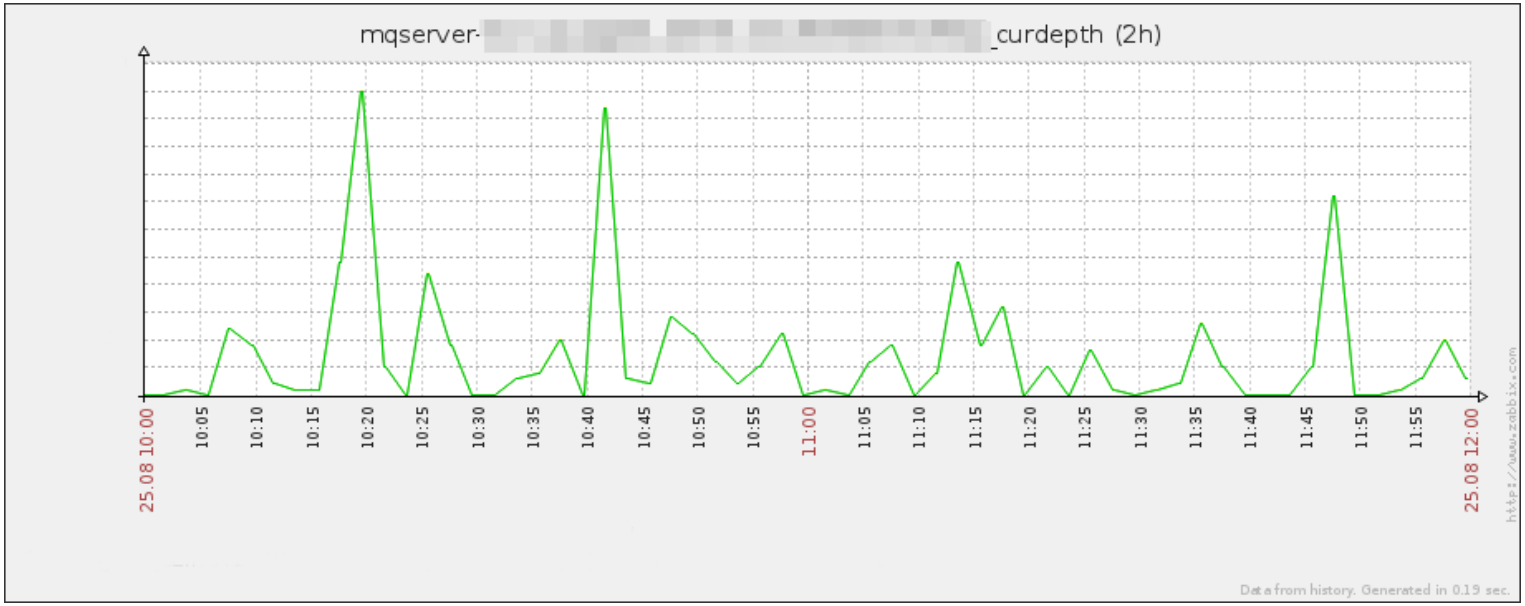
```
echo "display␣ql(QUEUE.NAME)␣curdepth" | runmqsc QMNAME
```

Queue capacity: max depth

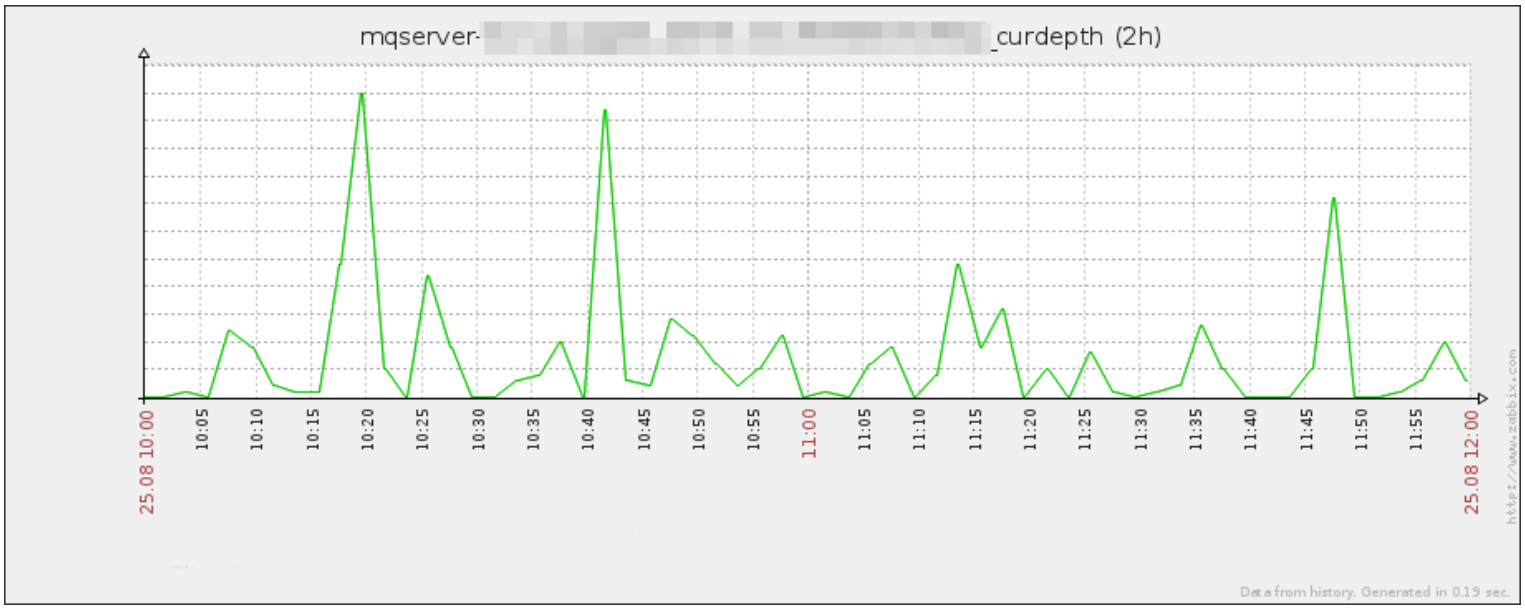
```
echo "display␣ql(QUEUE.NAME)␣maxdepth" | runmqsc QMNAME
```

Combined, they give us another key health check – queue % fill.

MESSAGE FLOW



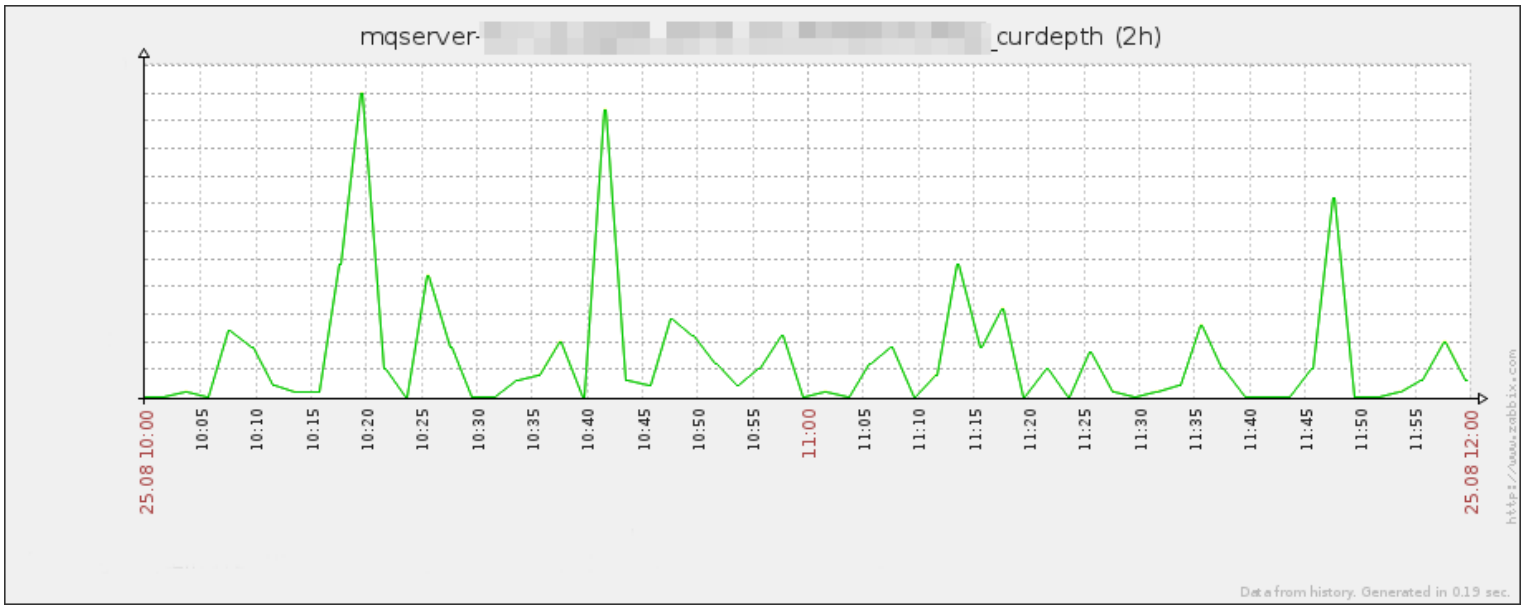
MESSAGE FLOW



First attempt at a trigger expression:

```
{queue_curdepth_itemkey.min(#20)} > 0
```

MESSAGE FLOW

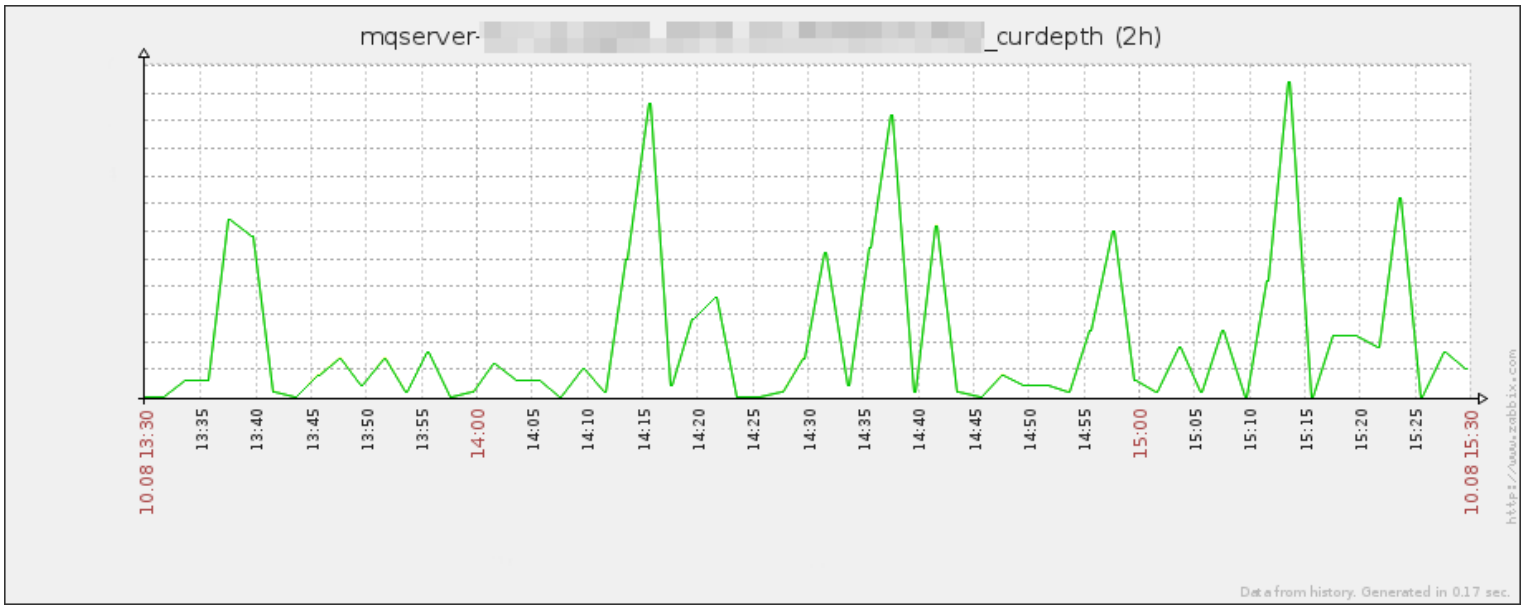


First attempt at a trigger expression:

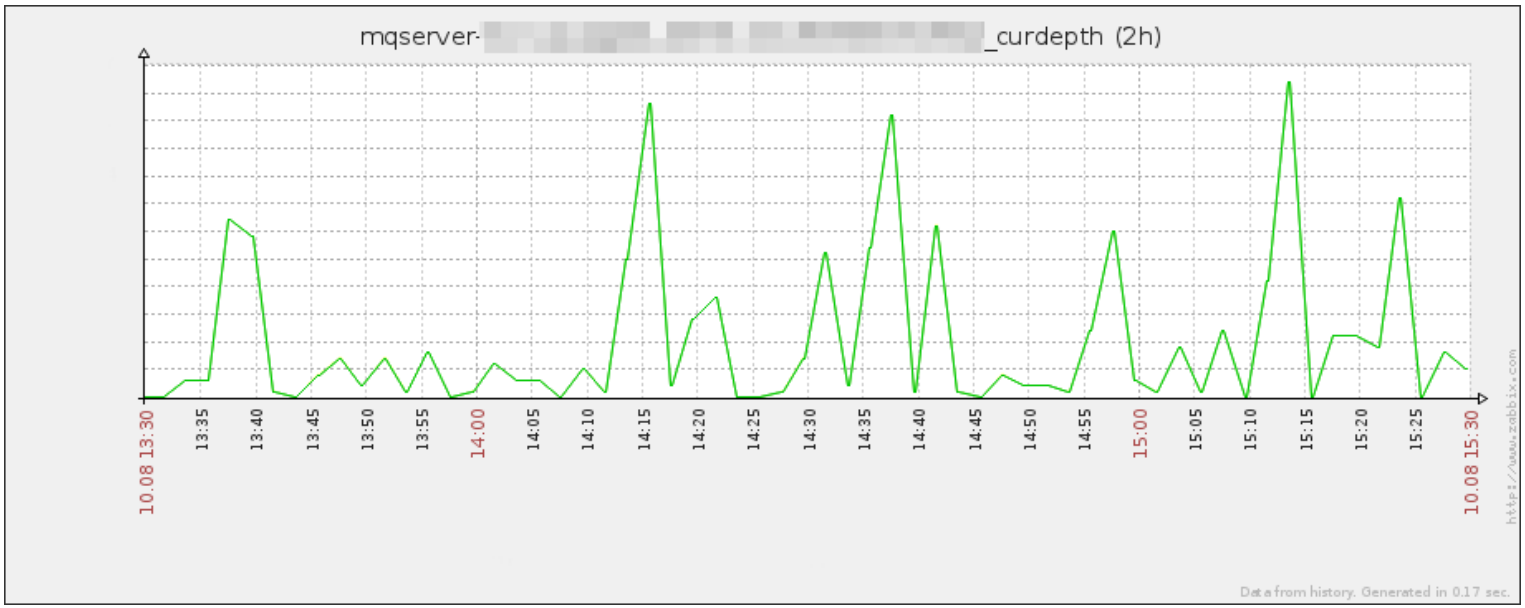
```
{queue_curdepth_itemkey.min(#20)} > 0
```

This is probably going to be insufficient.

MESSAGE FLOW, CONTINUED



MESSAGE FLOW, CONTINUED



To look for the drops over a period of time, we need to add a coupled calculated item, with the following formula:

```
last(queue_depth, #1) - last(queue_depth, #2)
```

MESSAGE FLOW – FINAL

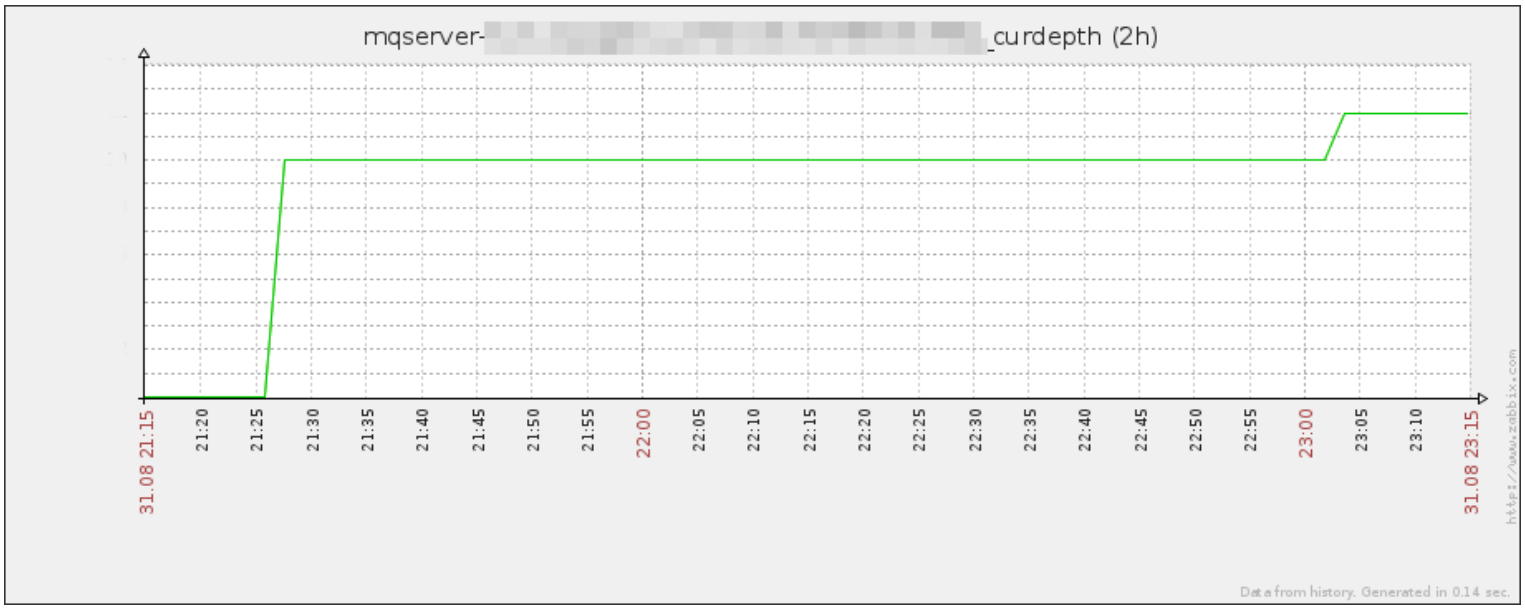
Now, we can examine if the queue is being unloaded, with this revised trigger expression:

```
{queue_depth.min(#20)} > 0 and {queue_depth_change.min(#20)} >= 0
```

MESSAGE FLOW – FINAL

Now, we can examine if the queue is being unloaded, with this revised trigger expression:

```
{queue_depth.min(#20)} > 0 and {queue_depth_change.min(#20)} >= 0
```



LOADABLE MODULES

- Support for loadable modules introduced in 2.2 (2013)
- Very easy to wrap around existing C code
- Very easy to get started (src/modules/dummy)
- Noticeable performance gains

...EXISTING C CODE

Look at the IBM provided samples, in this case, **amqsailq.c**.

```
/* bag for inquiry */
mqCreateBag(MQCBO_ADMIN_BAG, &adminBag, &cc, &r);
/* bag for the response */
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &cc, &r);

/* create the inquiry by putting request items in the bag */
mqAddString(adminBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, queue, &cc, &r);
mqAddInteger(adminBag, MQIA_Q_TYPE, MQQT_LOCAL, &cc, &r);
mqAddInquiry(adminBag, MQIA_CURRENT_Q_DEPTH, &cc, &r);

/* execute the inquiry */
mqExecute(
    /* needed params omitted here */
);
if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE) {
    MQDISC(&hConn, &cc, &r);
    exit(98);
}
/* when mqExecute was successful */
if ( compCode == MQCC_OK ) {
    mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &qAttrsBag, &cc, &r);
    mqInquireInteger(qAttrsBag, MQIA_CURRENT_Q_DEPTH, MQIND_NONE, &qDepth, &cc, &r);
    printf("%d\n", qDepth);
}
```

VERY EASY TO WRAP AROUND...

Look at the samples: *zabbix – 2.2.10/src/modules/dummy – orig/*.
In our case, you need to add around 40 lines of code.

```
static ZBX_METRIC keys[] = {
  {"mq.q.curdepth", CF_HAVEPARAMS, mq_q_curdepth, "ECHO"},
  {NULL}
};
```

```
int mq_q_curdepth(AGENT_REQUEST *request, AGENT_RESULT *result) {
  char *param;

  if (request->nparam != 1) {
    SET_MSG_RESULT(result, strdup("You must give the queue name as parameter"));
    return SYSINFO_RET_FAIL;
  }
  param = get_rparam(request, 0);

  // code that sets gathered_value goes here
  SET_UI64_RESULT(result, gathered_value);
  return SYSINFO_RET_OK;
}
```

PERFORMANCE GAINS

A simple shell script that gathers queue depth:

```
echo "DIS␣QL($1)␣CURDEPTH" | runmqsc | grep "[C]URDEPTH(" | grep -o '[0-9]\+'
```

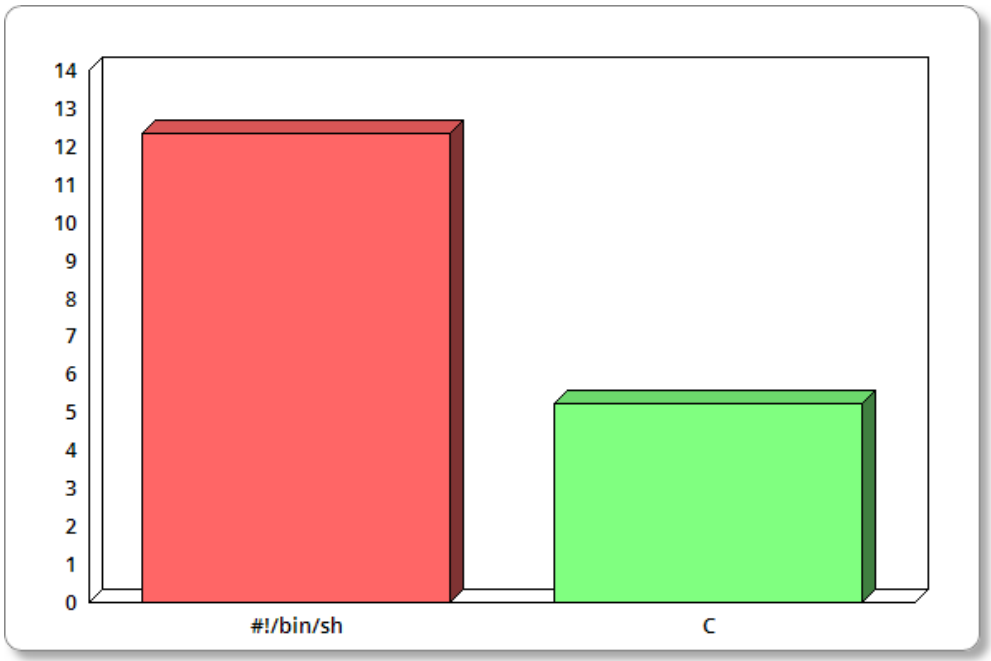
The same thing can be (quite easily) done in C.

PERFORMANCE GAINS

A simple shell script that gathers queue depth:

```
echo "DIS␣QL($1)␣CURDEPTH" | runmqsc | grep "[C]URDEPTH(" | grep -o '[0-9]\+'
```

The same thing can be (quite easily) done in C.



No surprises. Wrapping it in a module is even more effective.

SHARE

All the code that I've discussed here is uploaded to Zabbix Share – hope some of it proves useful.


```
https://share.zabbix.com/cat-app/queue-managers/ibm-mq-agent-module
```

Keep in mind that it's written as a proof of concept, for the purposes of this talk.

You should at least add additional error checking after each MQI call (see the MQCONN call for an example).

Any questions?

Thank you!