z/TPF Support For MongoDB Enhancements

Jamie Farmer

IBM **z/TPF**

April 3rd, 2017

Agenda

- A z/TPF administrator can define a user ID and password with associated MongoDB privileges using standard MongoDB tooling in a matter of seconds.
- A z/TPF administrator can dynamically enable MongoDB logging for z/TPF to diagnose problems, optimize client applications and provides an audit trail for MongoDB user management commands.
- A remote application developer can use the standard upsert option when updating a document to force z/TPF to create the document if it does not exist.
- A z/TPF database administrator can deploy more databases, specifically those without z/TPFDF default keys defined, for use through z/TPF support for MongoDB.

z/TPF Support For MongoDB User Security

A z/TPF administrator can define a user ID and password with associated MongoDB privileges using standard MongoDB tooling in a matter of seconds.

Current User Security For MongoDB

- The original z/TPF support for MongoDB user security was implemented through a user exit.
    - Customer was required to write user exit code to authenticate a user/password
- Authorization was handled by a rules based file that needed to be coded and loaded to z/TPF
- Although user security was available, it was time consuming and complex.

Enhanced MongoDB User Security

- User IDs and passwords can now be defined via z/TPF prime cras console command or through the standard remote MongoDB commands.
    - Passwords are safely persisted and encrypted on disk and in memory.
    - Persisted using the TPF file system (TFS)

- Creation of user IDs and passwords are immediately available for authentication on all processors in a loosely coupled complex.

Authorizing Users In MongoDB

- Authorization is based on Role Based Access Control (RBAC)
- A given user does not have access to any MongoDB data until a role providing access is assigned to that user.
    - Roles can be assigned via z/TPF prime cras console commands or through the standard remote MongoDB commands.

- Authorization through RBAC is identical to standard MongoDB providing synergy with z/TPF support for MongoDB.

Predefined Roles For Use

- The z/TPF system comes with built-in roles that can be assigned to defined users
  - read
    - Find (query) any document in any collections
  - readWrite
    - Find (query) any document in any collections
    - Insert, update, remove any document in any collection
  - userAdmin
    - Create/drop roles
    - Create/drop/update Users
    - Display users and roles

Same built-in roles are available in standard MongoDB

User Defined Roles

- What if you wanted to assign access to specific collections?
- Ability to define “user-defined roles”
  - Based on z/TPF MongoDB collection names and the actions that can be performed against it.

Let’s Look At A Few Examples…

Defining A User As The Administrator

Only a user with the “userAdmin” role can define users / roles remotely through MongoDB tools.

Jim is the user administrator for the system

- Prime Cras console commands have userAdmin

```
User: ZRUSR CREATE NAME-jim PWD-jimpwd ROLES-userAdmin Customdata-’User Administrator’

System: RUSR0002I 15.09.29 USER jim IS CREATED IN THE tpfdf DATABASE SUCCESSFULLY.
```

- Can do this remotely when server is defined without authentication enabled
  - Authentication / Authorization is bypassed when authentication disabled
  - --auth / --noauth on ZINET definition

Read Only Access To The Entire Database

Mary is a system administrator and allowed read-only access

Prime Cras Console Command:

```
User: ZRUSR CREATE NAME-mary PWD-marypwd ROLES-read CUSTOMDATA-'Read only access to DB'

System: RUSR0002I 15.09.29 USER mary IS CREATED IN THE tpfdf DATABASE SUCCESSFULLY.
```

Remote MongoDB Command (requires userAdmin authorization)

```
db.createUser({user: "mary",
               pwd: "marypwd",
               customData: {description: "Read only access to DB"},
               roles: ["read"]})
```

Creating User Defined Roles

- You can create user defined roles when the built-in roles cannot describe the privileges for a given user.
- A user defined role consists of a resource and the actions allowed against that resource
  - Resource: Database (tpfdf only), collection (z/TPFDF file) and tenant (subsystem user)
  - Actions: insert, update, find, remove

Examples Of User Defined Roles

- Let's say you have two MongoDB collections (z/TPF files)
  - PNR: Passenger Records
  - Flight: Collection containing Flight Information
- Role created allowing insert, update, remove and find privileges for PNR

```
db.createRole({ role: "PNRWrite",
                privileges: [ {
                   resource: { db: "tpfdf", collection: "PNR"},
                   actions: [ "find", "insert", "update", "remove"] }],
                roles: [] })
```

- Role created allowing find privileges for Flight

```
ZROLE CREATE NAME-FlightRead
ZROLE GRANT NAME-FlightRead COLL-Flight ACTION-find
```

Assigning User Defined Roles To Users

- Steve is a user that requires read-only access to flight

```
User:   ZRUSR CREATE NAME-steve PWD-stevepwd ROLES-FlightRead CUSTOMDATA-'Read only access to Flight'
System: RUSR0002I 15.09.29 USER steve IS CREATED IN THE tpfdf DATABASE SUCCESSFULLY.
```

- Now Steve, an existing user requires read-write access to PNR

```
db.updateUser("steve",
              {"roles": ["FlightRead", "PNRWrite"],
               "customData": {description: "Read access to Flight - ReadWrite to PNR" }
              })
```

- Steve is not allowed to look at any other collections than Flight and PNR

Displaying Users In the User Security Database

```
User:   ZRUSR DISPLAY NAME-*
System: RUSR0005I 13.55.38 DISPLAY USER * IN THE tpfdf DATABASE
        USER                              ROLES
        ------------------------------- ------------------------------------
        jim                               'userAdmin'
        mary                              'read'
        steve                             'FlightRead'  'PNRWrite'
        END OF DISPLAY
```

```
User:   ZRUSR DISPLAY NAME-mary
System: RUSR0006I 11.12.54 DISPLAY USER mary IN THE tpfdf DATABASE
           CustomData- 'Read only access to DB'
           Roles- 'read'
        END OF DISPLAY
```

Displaying Users and Roles From Standard Remote MongoDB Tooling

- Displaying users and roles has the same look and feel as standard tooling communicating with standard MongoDB tooling.
- Support the following standard MongoDB user/role display commands
  - db.getUsers() ← Retrieves all users
  - db.getUser(username) ← Retrieves single user
  - db.getRoles() ← Retrieves all roles
  - db.getRole(rolename) ← Retrieves single role

Dropping Users and Roles

- Ability to remove users and roles from the database

Prime Cras Console Command:

```
ZRUSR DROP NAME-steve
ZROLE DROP NAME-PNRWrite
```

Remote MongoDB Command (requires userAdmin authorization)

```
db.dropUser("steve")
db.dropRole("PNRWrite")
```

Setting Up And Managing The User Security Database

- z/TPF Keystore must be enabled and a pre-defined user security key must be created
  - The user database is encrypted using the z/TPF secure symmetric keystore
- You can dynamically change the encryption key used to encrypt the user security database
  - Transparent to running MongoDB applications
- Backup and Restore capability of the user security database

z/TPF MongoDB User Security Summary

- Defining users, their passwords and what they are allowed to access through z/TPF MongoDB is now much easier and can be done using standard MongoDB tooling.

- PUT 13 APAR P43870 provides this support.

z/TPF Support For MongoDB Logging

A z/TPF administrator can dynamically enable MongoDB logging for z/TPF to diagnose problems, optimize client applications and provides an audit trail for MongoDB user management commands.

Diagnosing z/TPF MongoDB Problems

- Let’s say you are testing an new MongoDB application
  - How do you diagnose problems that may result from the new application?

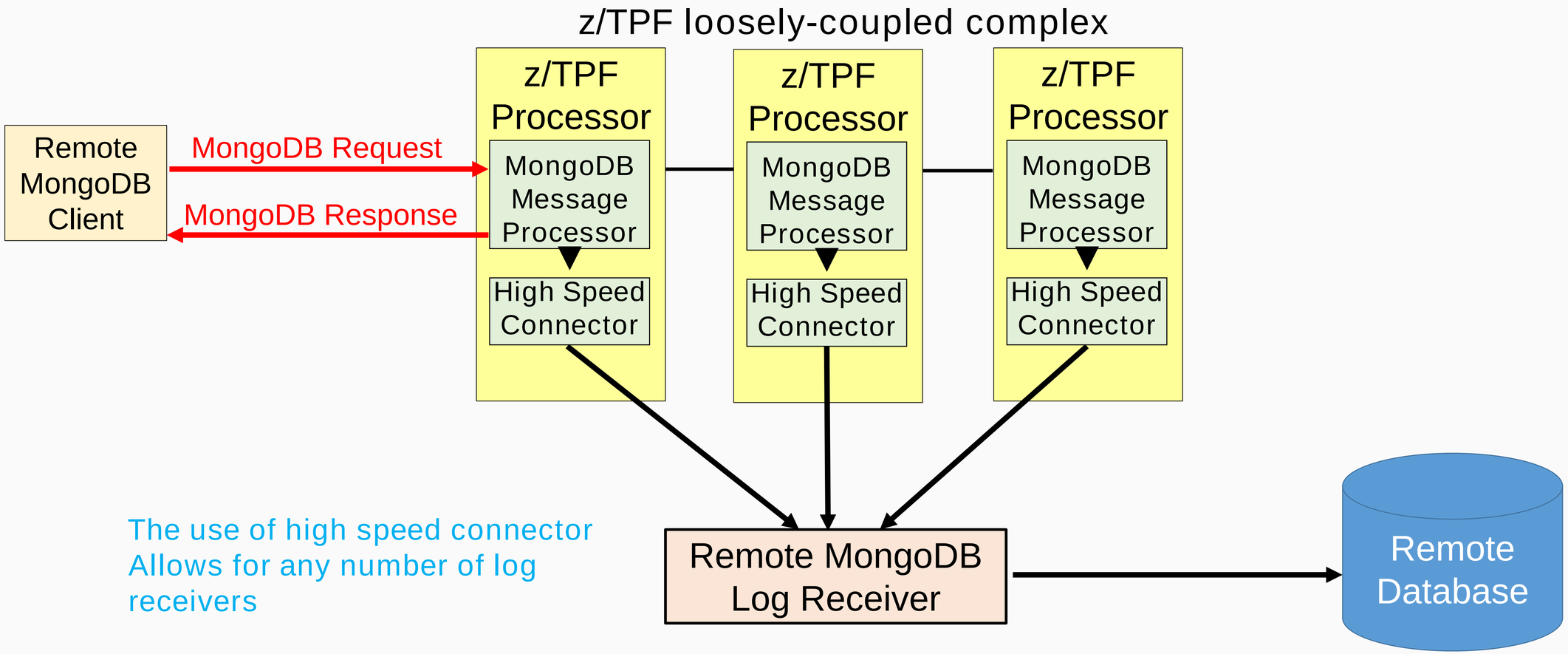
- Currently there is no logging available for MongoDB requests from remote users.
  - Can use IP trace to help, but its very difficult to pinpoint the problematic request.

z/TPF Support for MongoDB Logging

- Can enable logging of MongoDB client requests and z/TPF responses dynamically.
  - You can customize what requests you want logged based on…
    - Type of request (insert, update, find, remove)
    - Collection (z/TPFDF file) being accessed
  - Can optionally include the z/TPFDF binary data that was accessed during the operation.

- Uses z/TPF's High Speed Connector as the transport mechanism for log data.
- Transfers the log data in BSON format (binary JSON) for easy consumption by remote log receivers.

z/TPF Support for MongoDB Logging Architecture



The use of high speed connector
Allows for any number of log receivers

Selecting A Log Receiver

- Log data is sent through the high speed connector interface.
- A given log message consists of
  - High Speed Connector Header (containing the length of the data).
  - MongoDB Log message in BSON format.
- A log receiver can be home-grown server code or can be open source logging packages (ie. LogStash)
  - The z/TPF lab used LogStash during development
  - Open source logging packages would require slight modifications
    - Stripping off the high speed connector header
    - Possibly creating a BSON plugin for the open source package

Customizing Logging For z/TPF MongoDB

- Using the new ZMONG LOG SET command you can indicate what collections and what actions against those collections should be logged.

Sets logging of insert, update and remove operations against collection PNR

```
User:   ZMONG LOG SET COLLECTION-PNR ACTION-insert,update,remove
System: CMNG0109I 13.15.17 LOGGING ACTIONS FOR COLLECTION-PNR IN DATABASE-tpfdf ARE SET.
           ACTION-insert,update,remove
           BINARY-NO
        END OF DISPLAY
```

Sets logging for all MongoDB operations against collection Flight with the binary option enabled

```
User:   ZMONG LOG SET COLLECTION-Flight ACTION-insert,update,remove,find BINARY
System: CMNG0109I 13.18.17 LOGGING ACTIONS FOR COLLECTION-Flight IN DATABASE-tpfdf ARE SET.
           ACTION-insert,update,remove,find
           BINARY-YES
        END OF DISPLAY
```

Starting MongoDB Logging

- z/TPF MongoDB logging is started for an individual MongoDB server running on z/TPF.

Starts MongoDB logging for the server named MONGO using the default group name of IMONGLOG

```
User:   ZMONG LOG START S-MONGO
System: CMNG0106I 13.19.19 LOGGING FOR z/TPF SUPPORT FOR MONGODB ON SERVER-MONGO IS STARTED
```

Starts MongoDB logging for the server named MONGO using group name ITESTLOG

```
User:   ZMONG LOG START S-MONGO EPT-ITESTLOG
System: CMNG0106I 13.20.20 LOGGING FOR z/TPF SUPPORT FOR MONGODB ON SERVER-MONGO IS STARTED
```

z/TPF MongoDB Logging Summary

- Dynamically enable logging by collection and action
- Designed for production use
    - Each MongoDB request logged adds about 10% overhead for that request
        - With BINARY option enabled adds about 15%
        - Your results may vary

- PUT 13 APAR PJ44239 provides this support.

z/TPF Support For MongoDB upsert Option

A remote application developer can use the standard upsert option when updating a document to force z/TPF to create the document if it does not exist.

What is the MongoDB “upsert” option

- In MongoDB upsert is an option supplied on an update request
  - If document to update is found, the update is made to the existing document
  - If document is not found, then it is created
- Useful for replicating data into a server
- Often used by standard frameworks when interfacing with a MongoDB server
- Can be accomplished today by interrogating whether a document exists
  - If it does not exist, then issue an insert, otherwise issue an update
  - Requires more MongoDB flows and more complicated MongoDB client logic

MongoDB “upsert” Option For z/TPF

- Can now issue the standard upsert option on update requests sent to z/TPF.
- The z/TPF system will honor the update request and insert a new document if the document does not exist.

- PUT 13 APAR PJ43870 provides this support.

Disclaimer

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

Lifting The Requirement For z/TPFDF Default Keys

A z/TPF database administrator can deploy more databases, specifically those without z/TPFDF default keys defined, for use through z/TPF support for MongoDB.

Updating The Database In MongoDB Requires z/TPFDF Default Keys

- Currently, to perform any update operation (insert, update, remove), z/TPFDF file being accessed must have default keys defined in the DBDEF.
  - MongoDB client have no knowledge of how logical records are organized in a subfile (You don’t want them to!)

- Databases that do not have default keys defined cannot be deployed for use by MongoDB for update
  - Many customers do not want to change their z/TPFDF DBDEF definitions.

Lifting The z/TPFDF Default Key Restriction

- z/TPF MongoDB is looking at lifting this restriction.
- User can supply MongoDB specific configuration to let the system know how a z/TPFDF file is organized.
- Today, the MongoDB configuration requires you to tell the system the logical records contained in a z/TPFDF file
  - This has been expanded to optionally define organization

```
<tns:Lrec name="PassengerNumberRecord" id="70" unique="true">
    <tns:Organization>
        <!-- PassengerNumberRecord is organized by passenger number in ascending order. -->
        <tns:Key field="Number" org="Ascending"/>
    </tns:Organization>

</tns:Lrec>
```

Lifting The z/TPFDF Default Key Restriction Summary

- The number of databases that can be deployed for use with MongoDB increases by lifting this restriction.

- APAR PJ44378 is currently in development.

Summary

- PJ43870 (PUT 13)
  - A z/TPF administrator can define a user ID and password with associated MongoDB privileges using standard MongoDB tooling in a matter of seconds.
- PJ44239 (PUT 13)
  - A z/TPF administrator can dynamically enable MongoDB logging for z/TPF to diagnose problems, optimize client applications and provide an audit trail for MongoDB user management commands.
- PJ43870 (PUT 13)
  - A remote application developer can use the standard upsert option when updating a document to force z/TPF to create the document if it does not exist.
- PJ44378 (PUT 14)
  - A z/TPF database administrator can deploy more databases, specifically those without z/TPFDF default keys defined, for use through z/TPF support for MongoDB.

THANK YOU

Questions or comments?

Jamie Farmer

IBM **z/TPF**

April 3rd, 2017

Notes

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.

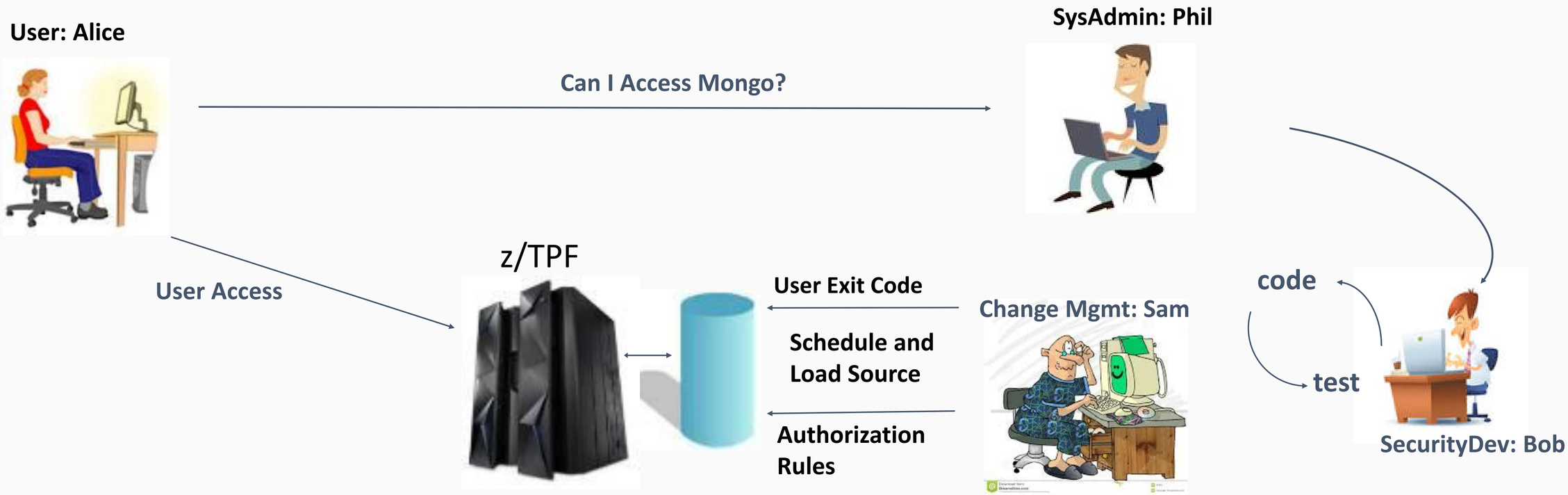
Backup Slides

z/TPF Support For MongoDB User Security

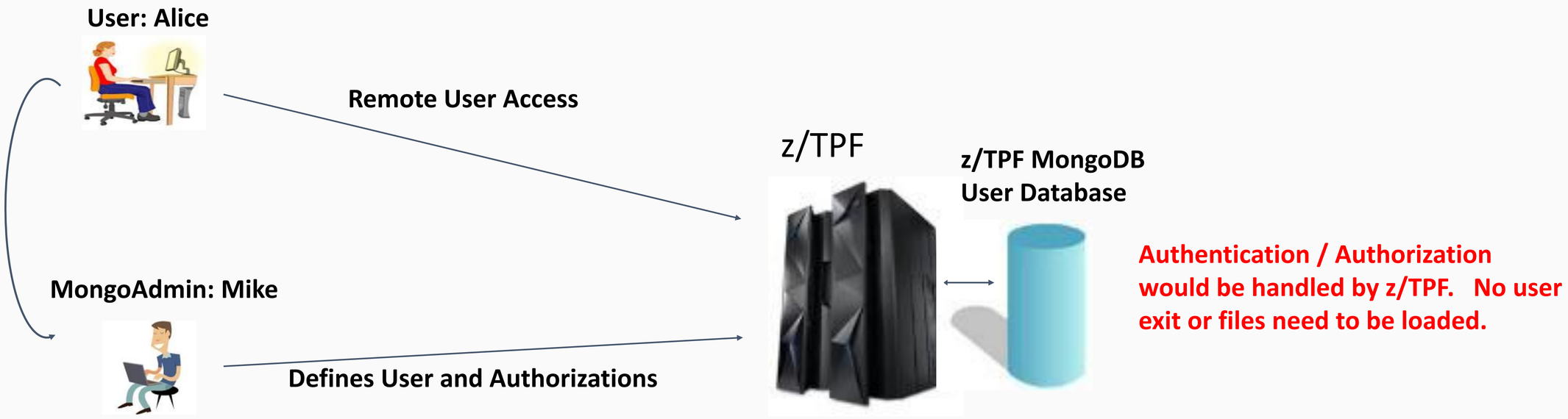
A z/TPF administrator can define a user ID and password with associated MongoDB privileges using standard MongoDB tooling in a matter of seconds.

Adding A User For MongoDB Access

User “Alice” wants access to the z/TPF MongoDB Server



Adding A User For MongoDB Access



Displaying Roles In the User Security Database

```
User:   ZROLE DISPLAY NAME-*
System: ROLE0006I 10.37.59 DISPLAY ROLE * FROM THE tpfdf DATABASE
        BUILTIN ROLE
        ------- --------------------------------
                FlightRead
                PNRWrite
          YES   read
          YES   readWrite
          YES   userAdmin
        END OF DISPLAY


User:   ZROLE DISPLAY NAME-PNRWrite
System: ROLE0007I 15.39.52 DISPLAY ROLE PNRWrite IN THE tpfdf DATABASE
            BUILTIN-YES
            PRIVILEGES-
               RDB-tpfdf, COLLECTION-PNR, TENANT-*
                 ACTIONS-insert update remove find
        END OF DISPLAY
```

Setting Up The User Security Database

- The user database is encrypted using the z/TPF secure symmetric keystore
- Keystore must be enabled and a pre-defined user security key must be created

ZKEYS GENERATE ENC-IUSERSEC DEC-IUSERD1 CIPHER-AES256CBC NEW

* IUSERSEC is the predefined keyname needed for the user security database

Changing Encryption Keys In The User Security Database

- You can dynamically change the encryption key used to encrypt the user security database
- This change is seem less with NO impact to active z/TPF MongoDB applications.

```
ZKEYS GENERATE ENC-IUSERSEC DEC-IUSERD2 CIPHER-AES256CBC

ZKEYS ACTIVATE ENC-IUSERSEC DEC-IUSERD2
```

- The system automatically detects the new IUSERSEC and re-encrypts the database on file and in memory (on all processors in a loosely-coupled complex)

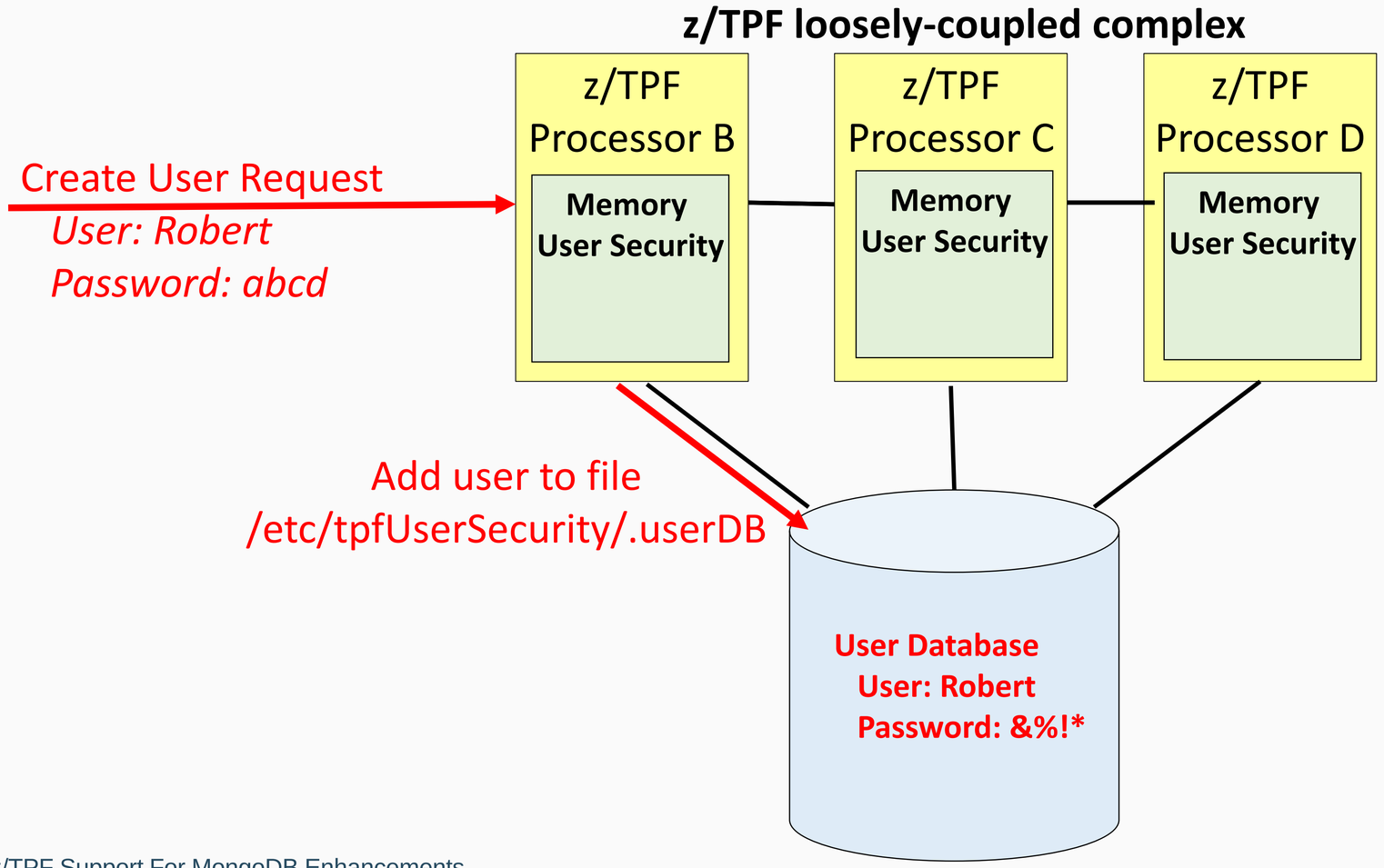
```
CXUS0002I 08.55.18 THE USER SECURITY DATABASE WAS RE-ENCRYPTED SUCCESSFULLY

CXUS0001I 08.55.18 MEMORY FOR THE USER SECURITY DATABASE WAS REFRESHED
```

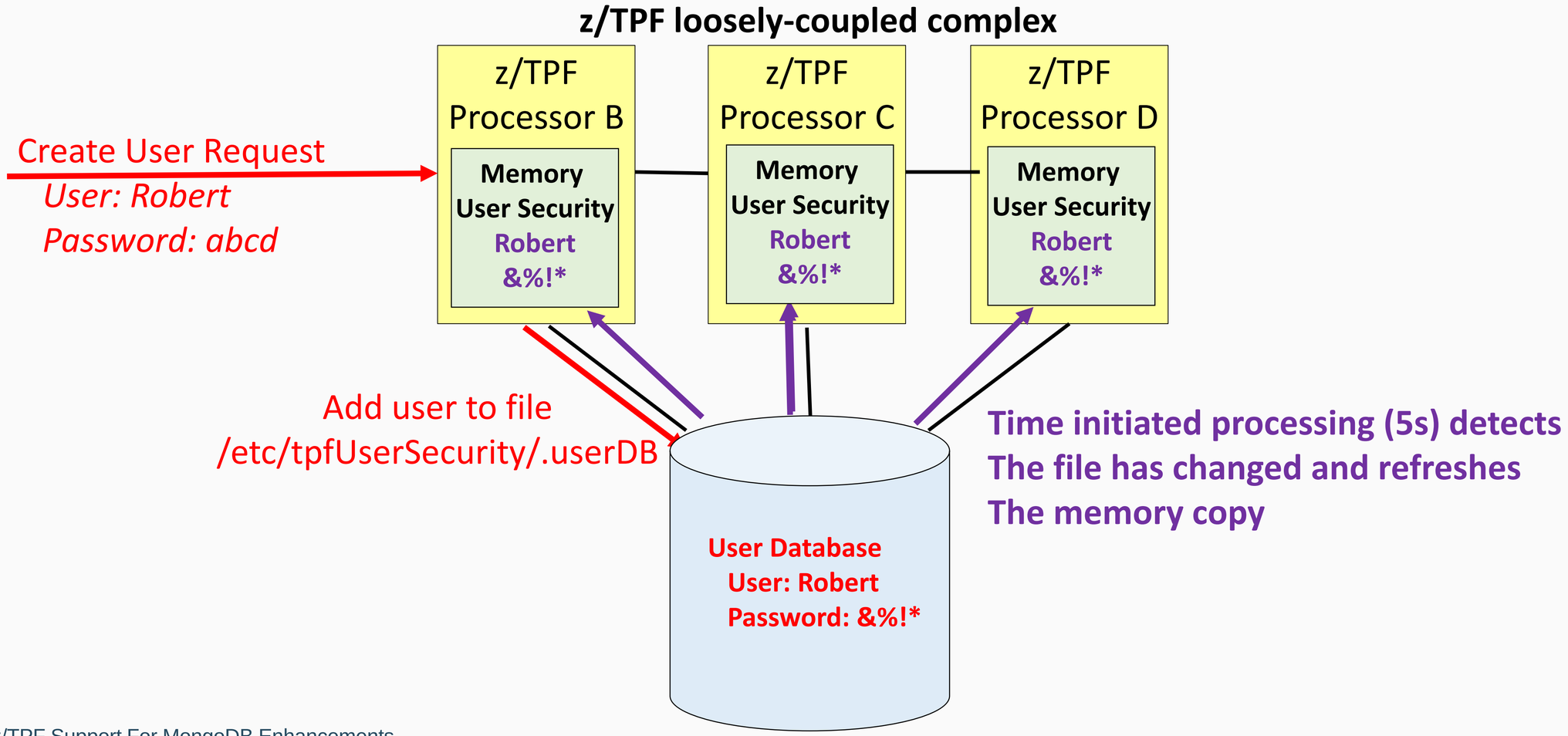
Backup And Restore The User Security Database

- The user security database persistence is a file on the file system
  - /etc/tpfUserSecurity/.userDB
  - to active z/TPF MongoDB applications.
- Backing up the user security database is to FTP the file off of TPF
- Restoring the user security database is to FTP the file back to TPF
  - System will automatically detect the new file and refresh the memory version of the user database (on all loosely-coupled processors)

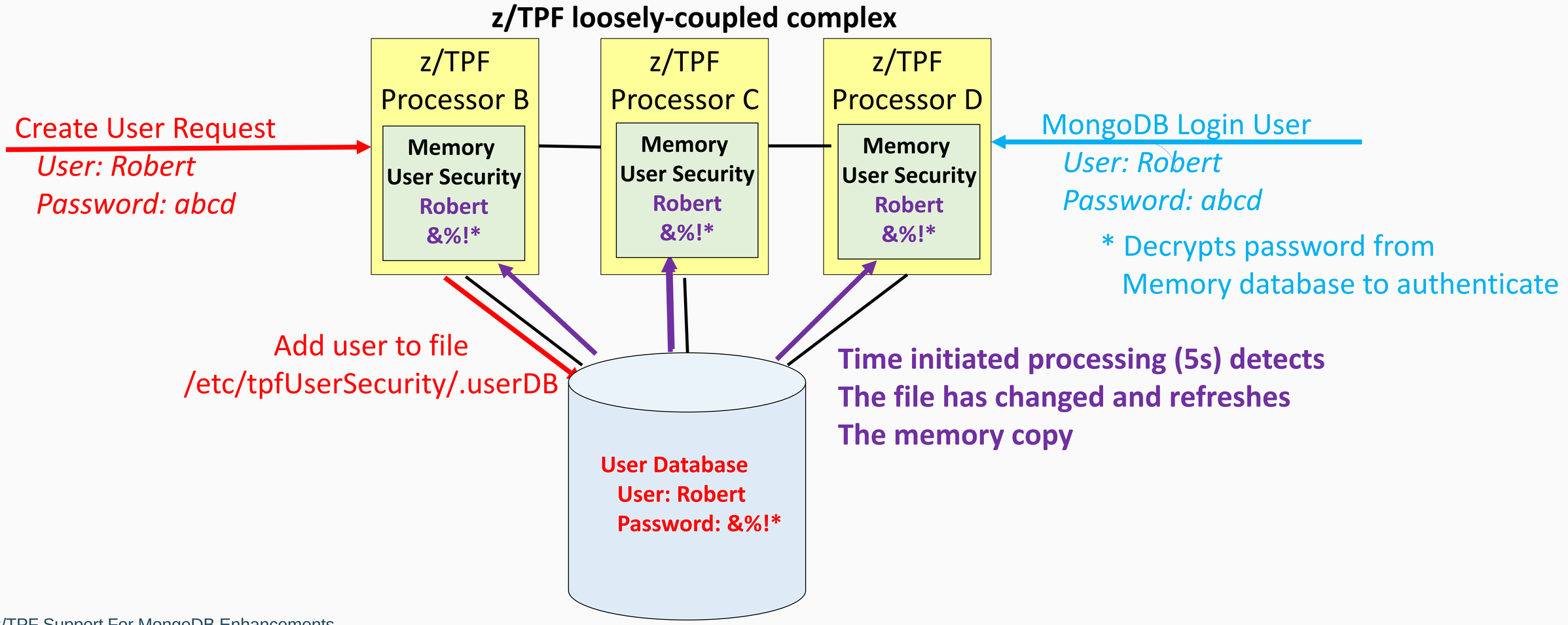
z/TPF User Security Architecture – Creating A User



z/TPF User Security Architecture – Refresh Memory



z/TPF User Security Architecture – Login New User



z/TPF Support For MongoDB Logging

A z/TPF administrator can dynamically enable MongoDB logging for z/TPF to diagnose problems, optimize client applications and provides an audit trail for MongoDB user management commands.

Creating The Endpoint Group Definition For MongoDB Logging

- The high speed connector group of endpoints must be created and loaded before log data can be sent.

```
<tns:EndpointGroup>
  <tns:Endpoint>
    <tns:endpointName>MLOGEPT1</tns:endpointName>
    <tns:role>PRIMARY</tns:role>
    <tns:destination>9.57.9.171:4514</tns:destination>
    <tns:startSocket>1</tns:startSocket>
    <tns:maxSocket>10</tns:maxSocket>
  </tns:Endpoint>
  <tns:groupName>IMONGLOG</tns:groupName>
  <tns:qMaxDepth>0</tns:qMaxDepth>
  <tns:qThreshold>45</tns:qThreshold>
  <tns:syncTimeout>500</tns:syncTimeout>
  <tns:heartbeatInterval>0</tns:heartbeatInterval>

</tns:EndpointGroup>
```

MongoDB Log Message Format – Header

```
"header":{
   "procId":"B",
   "tenant":"HPN ",
   "user":"Mike",
   "reqAct":"insert",
   "reqTime":"19:17:06.282670",
   "reqDur":1560, "tpfHost:”9.57.13.99”,
   "userIP":"9.12.80.181",
   "lPort":27017,
   "rPort":61808,
   "algLen":0,
   "algStr": {"$binary":"","$type":"generic"},
   "objId":{"$oid":"000000000000000180e9b14"}

}
```

MongoDB Log Message Format – Request

```
"req":{
  "type":"query",
  "fullCollectionName":
  "tpfdf.$cmd", "query":{
    "insert":"PNR",
    "documents":[
      {
        "_id":{"$oid":"57a8cd0a2553fabcf645bf08"},
        "_index":{"PnrByName":{"name":"ZZSMITH"}},
                  "PassengerNumberRecord":[{"PassengerNumber":100}]
                  "AddressRecord":[{"Address":"267 Jordan Blvd, LasVegas, Nevada 58521"}],
                  "FlightHistoryRecord":[ {
                     "Spare":"\u0000",
                     "FlightInfo":{
                       "Flight":{
                         "FlightDate":1.0,
                         "FlightNumber":"10"
                       },
                       "Origin":"ALD",
                       "Destination":"TDE"
                     }
                  } ]
      } ],
    "ordered":true} }
```

MongoDB Log Message Format – Binary

You can optionally include the binary data associated with the specific z/TPFDF operations

```
"binary":[
     {
      "operation":"INSERT",
      "dataLength":28,
      "data":{"$binary":"001c80e9e9e2d4c9e3c840404040404040404040404040404040404040","$type":"generic"}
     },
     {
      "operation":"INSERT",
      "dataLength":28,
      "data":{"$binary":"001c80e9e9e2d4c4e3c940404040404040404040404040404040404040","$type":"generic"}
     }
}
```